Finding High-Quality Local Minima in Derivative-Free Optimization

Jeffrey Larson Stefan Wild

Argonne National Laboratory

October 20, 2015

Motivation

- We want to identify distinct, “high-quality”, local minimizers of

$$\begin{aligned} \text{minimize } & f(x) \\ l \leq x & \leq u \\ x & \in \mathbb{R}^n \end{aligned}$$

- High-quality can be measured by more than the objective.

Motivation

- We want to identify distinct, "high-quality", local minimizers of

$$\begin{aligned} \text{minimize } & f(x) \\ l \le x & \le u \\ x & \in \mathbb{R}^n \end{aligned}$$

- High-quality can be measured by more than the objective.
- Derivatives of f may or may not be available.

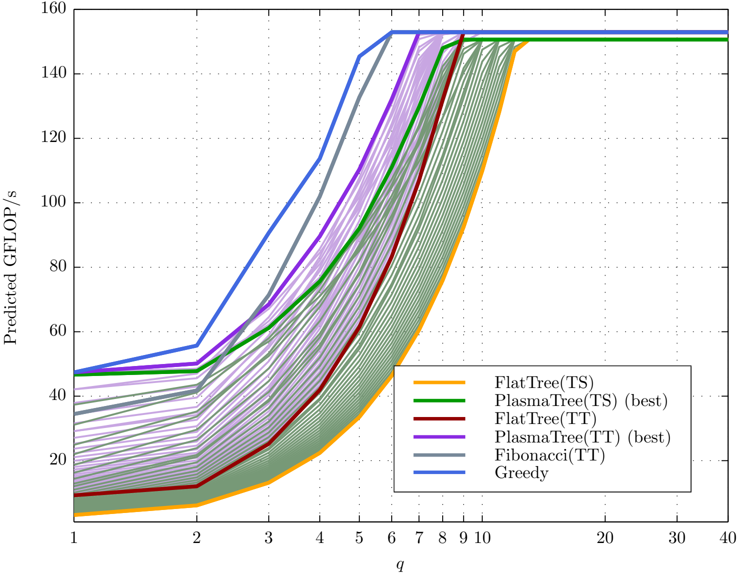
Motivation

- We want to identify distinct, "high-quality", local minimizers of

$$\begin{aligned} \text{minimize}\, f(x) \\ l \leq x \leq u \\ x \in \mathbb{R}^n \end{aligned}$$

- High-quality can be measured by more than the objective.
- Derivatives of f may or may not be available.
- The simulation f is likely using parallel resources, but it does not utilize the entire machine.

Why concurrency? Tiled QR example



[Bouwmeester, et al., Tiled QR Factorization Algorithms, 2011]

Global optimization is difficult

Theorem (Törn and Žilinskas, *Global Optimization*, 1989)

An algorithm converges to the global minimum of any continuous f on a domain $\mathcal{D}$ if and only if the algorithm generates iterates that are dense in $\mathcal{D}$.

Global optimization is difficult

Theorem (Törn and Žilinskas, *Global Optimization*, 1989)

An algorithm converges to the global minimum of any continuous f on a domain $\mathcal{D}$ if and only if the algorithm generates iterates that are dense in $\mathcal{D}$.

- Either assume additional properties about the problem
  - convex f
  - separable f
  - finite domain $\mathcal{D}$

Global optimization is difficult

Theorem (Törn and Žilinskas, *Global Optimization*, 1989)

An algorithm converges to the global minimum of any continuous f on a domain $\mathcal{D}$ if and only if the algorithm generates iterates that are dense in $\mathcal{D}$.

- Either assume additional properties about the problem
  - convex f
  - separable f
  - finite domain $\mathcal{D}$
- Or possibly wait a long time (or forever)

Global optimization is difficult

Theorem (Törn and Žilinskas, *Global Optimization*, 1989)

An algorithm converges to the global minimum of any continuous f on a domain $\mathcal{D}$ if and only if the algorithm generates iterates that are dense in $\mathcal{D}$.

- Either assume additional properties about the problem
  - convex f
  - separable f
  - finite domain $\mathcal{D}$
  - concurrent evaluations of f
- Or possibly wait a long time (or forever)

Global optimization is difficult

Theorem (Törn and Žilinskas, *Global Optimization*, 1989)

An algorithm converges to the global minimum of any continuous f on a domain $\mathcal{D}$ if and only if the algorithm generates iterates that are dense in $\mathcal{D}$.

- Either assume additional properties about the problem
  - convex f
  - separable f
  - finite domain $\mathcal{D}$
  - concurrent evaluations of f
- Or possibly wait a long time (or forever)

The theory can be more than merely checking that a method generates iterates which are dense in the domain.

Global optimization is difficult

Theorem (Törn and Žilinskas, *Global Optimization*, 1989)

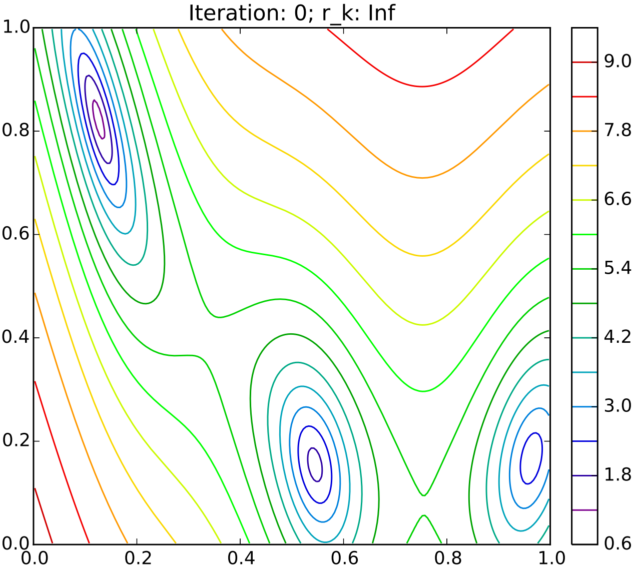
An algorithm converges to the global minimum of any continuous f on a domain $\mathcal{D}$ if and only if the algorithm generates iterates that are dense in $\mathcal{D}$.

- Either assume additional properties about the problem
  - convex f
  - separable f
  - finite domain $\mathcal{D}$
  - concurrent evaluations of f
- Or possibly wait a long time (or forever)

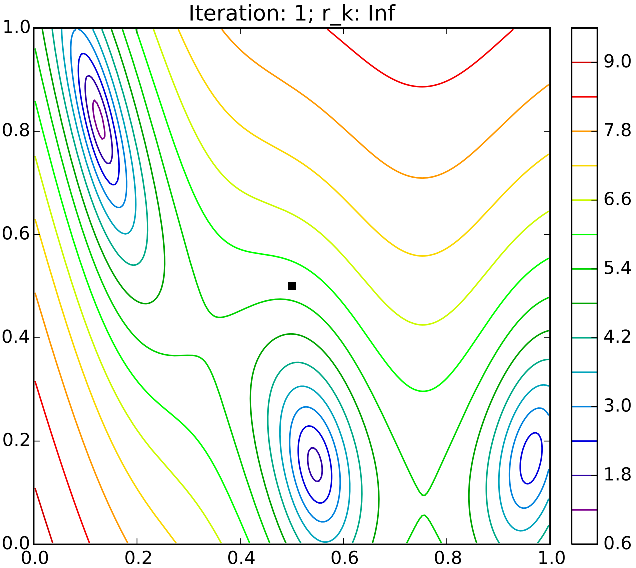
The theory can be more than merely checking that a method generates iterates which are dense in the domain.

An algorithm must trade-off between "refinement" and "exploration".

DIRECT

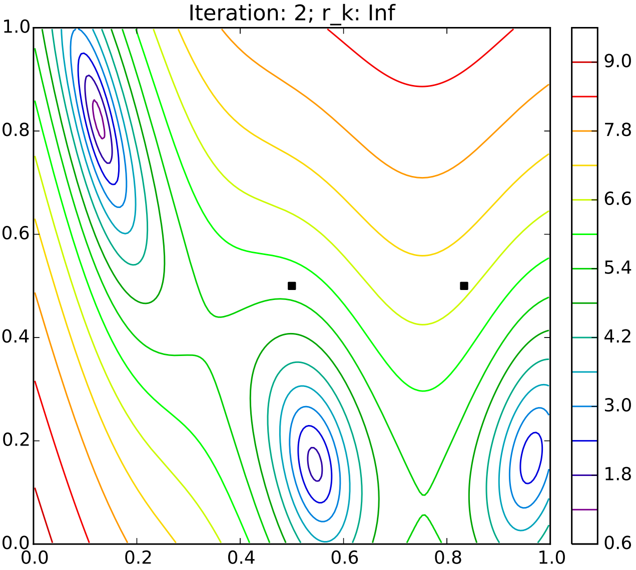

Iteration: 0; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



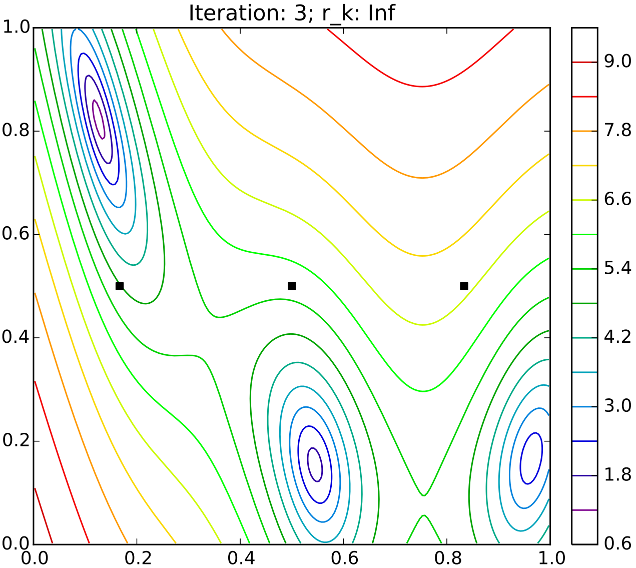
Iteration: 1; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



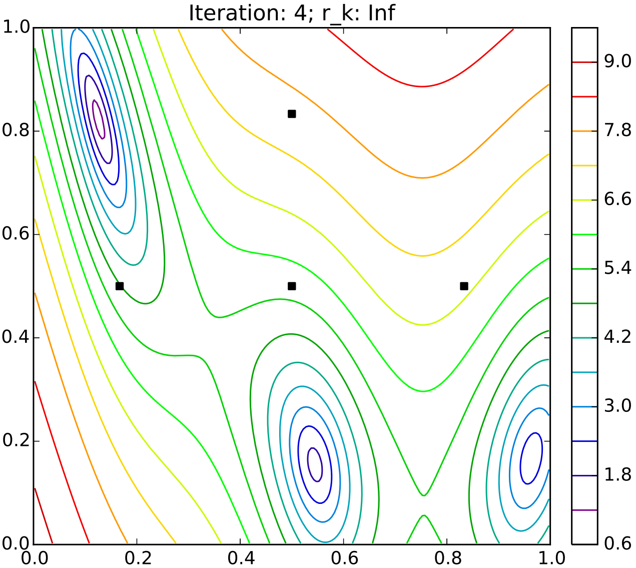
Iteration: 2; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



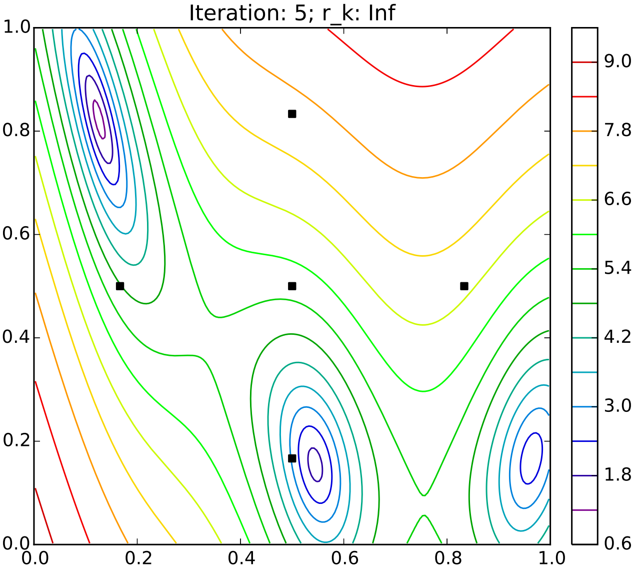
Iteration: 3; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



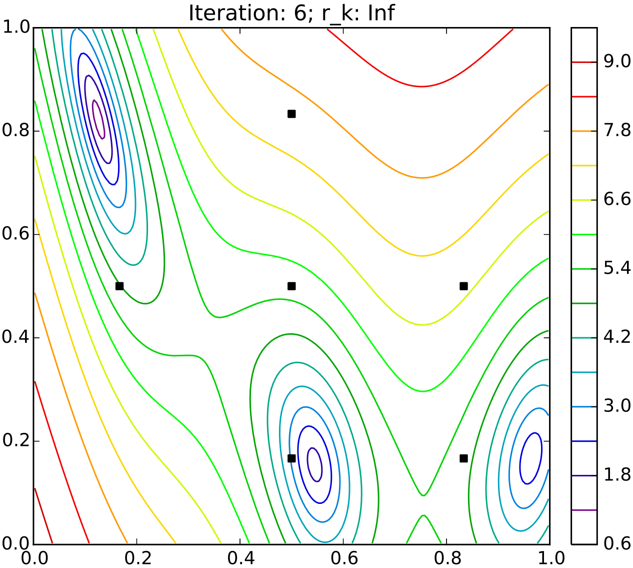
Iteration: 4; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



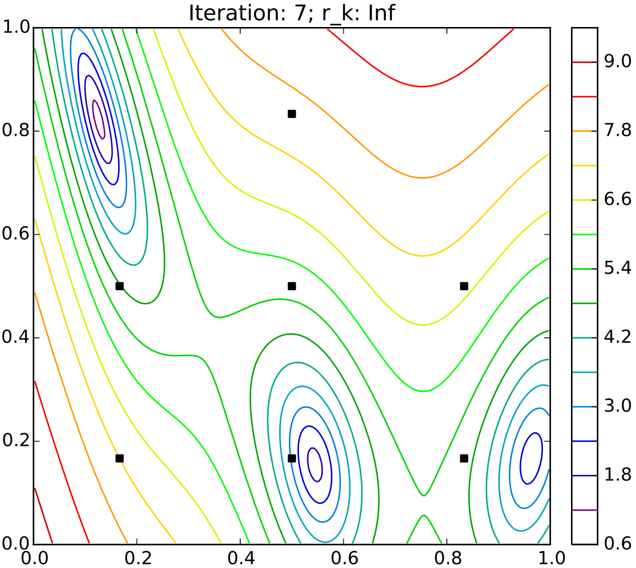
Iteration: 5; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



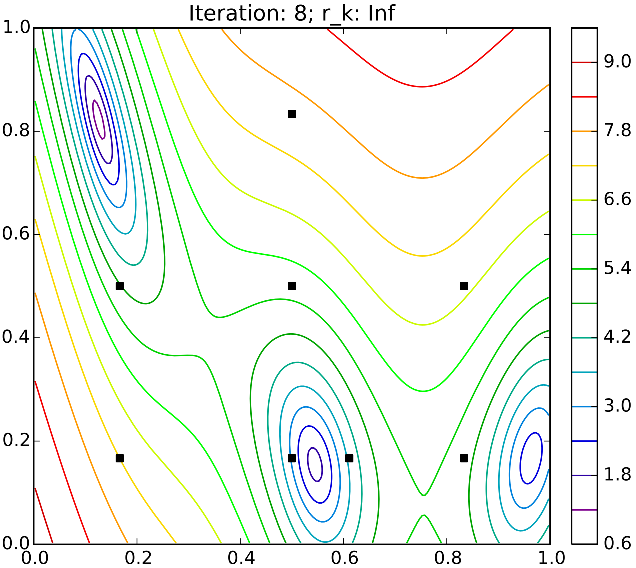
Iteration: 6; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



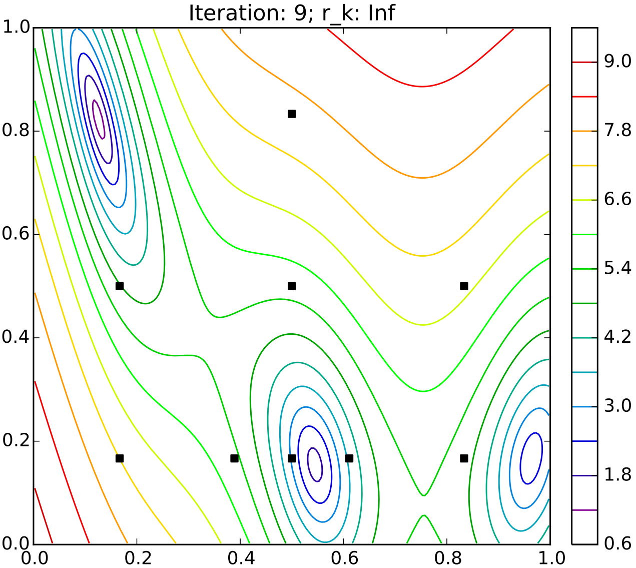
Iteration: 7; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



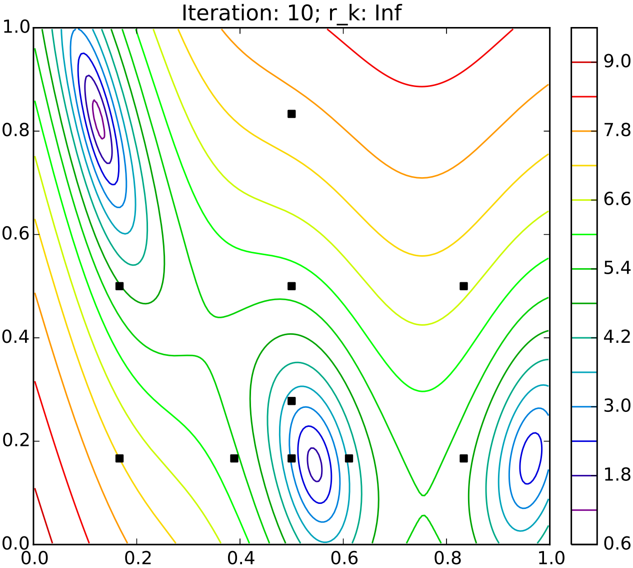
Iteration: 8; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



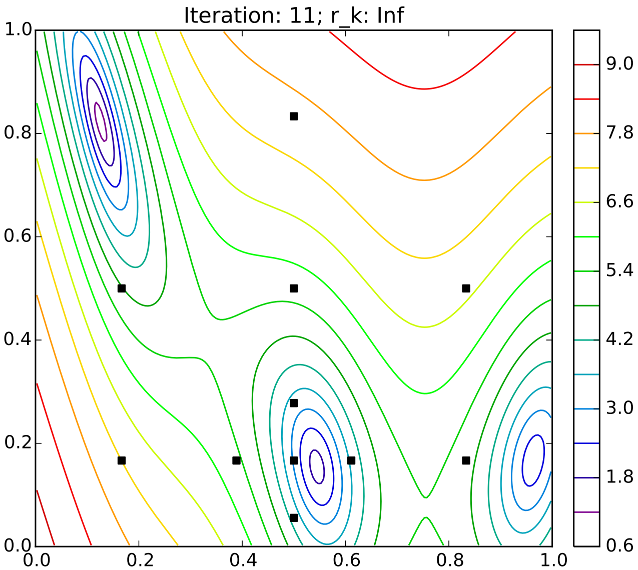
Iteration: 9; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



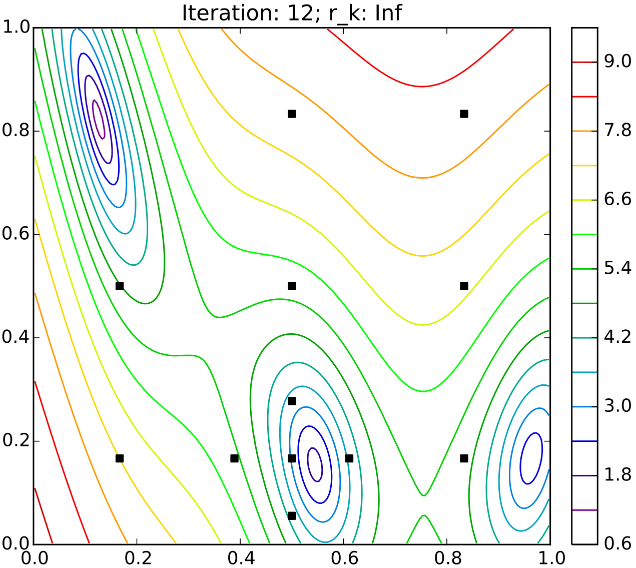
Iteration: 10; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



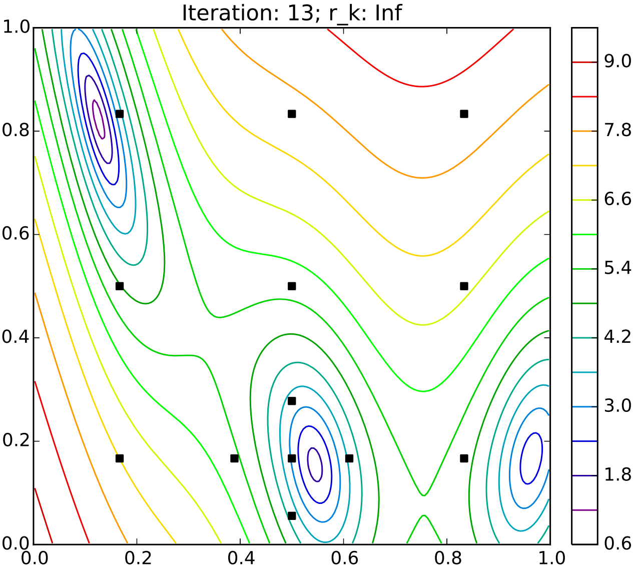
Iteration: 11; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



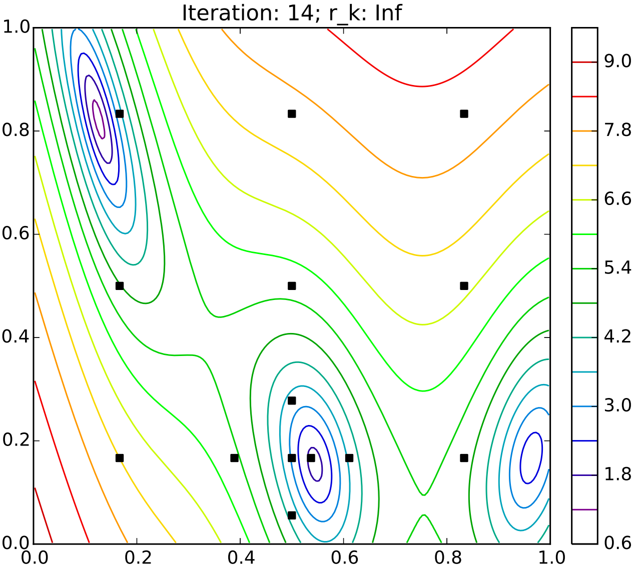
Iteration: 12; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



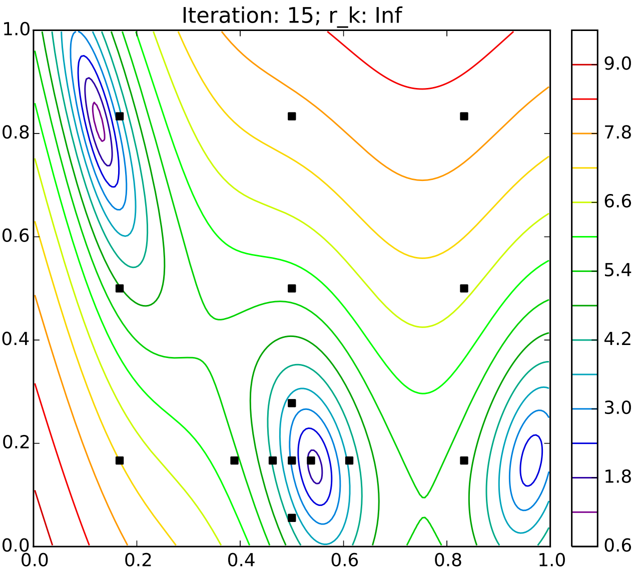
Iteration: 13; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



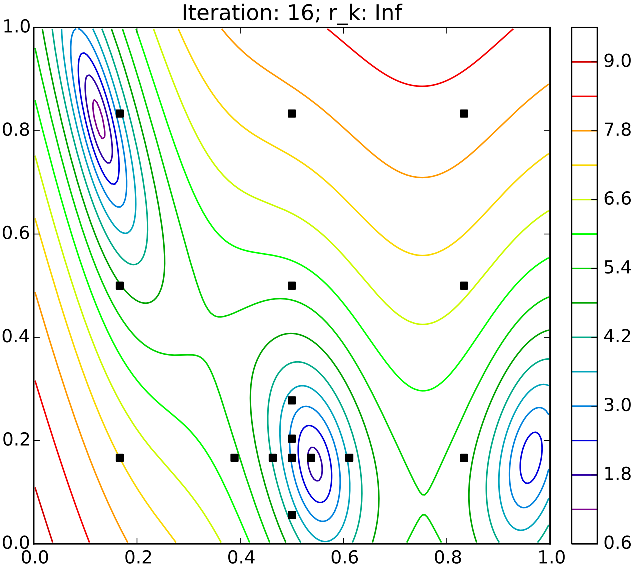
Iteration: 14; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



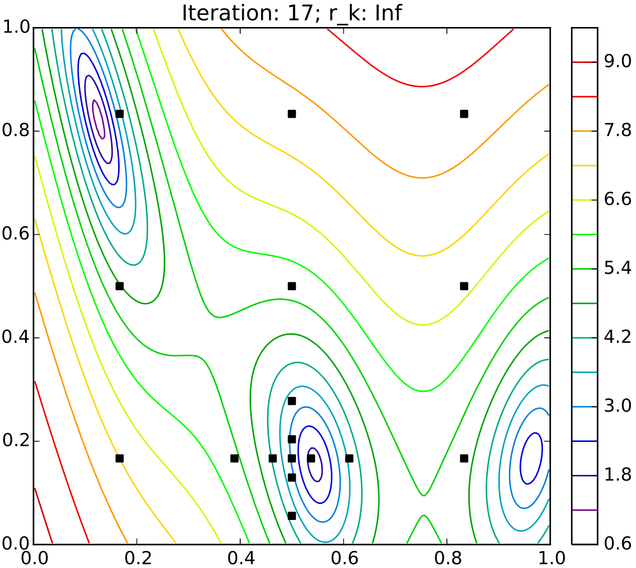
Iteration: 15; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



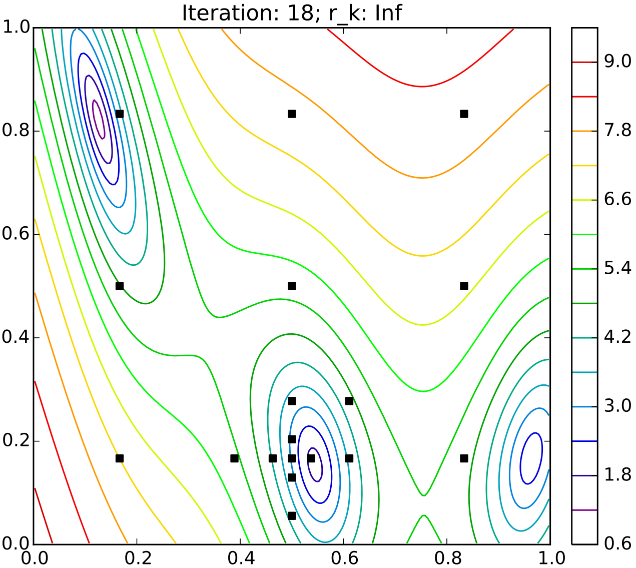
Iteration: 16; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



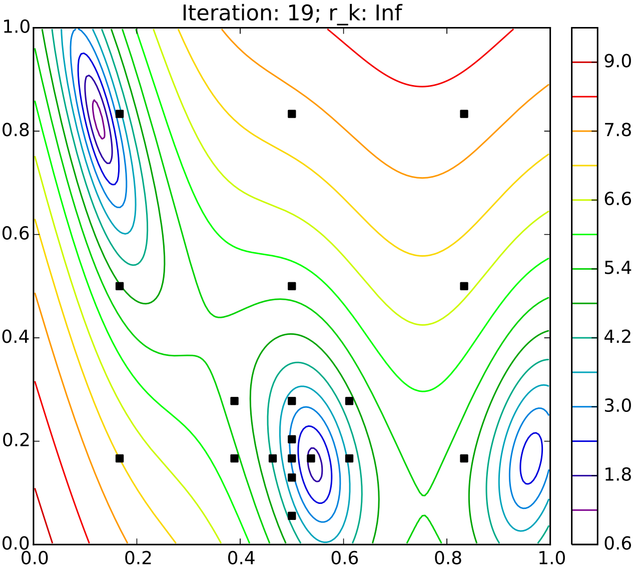
Iteration: 17; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



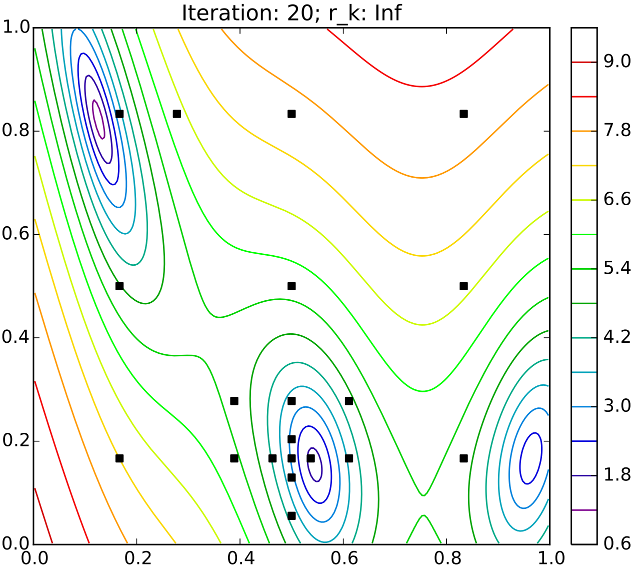
Iteration: 18; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



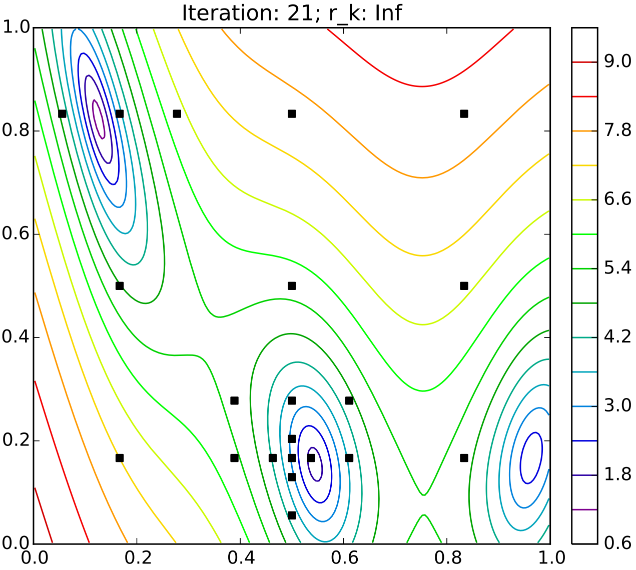
Iteration: 19; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



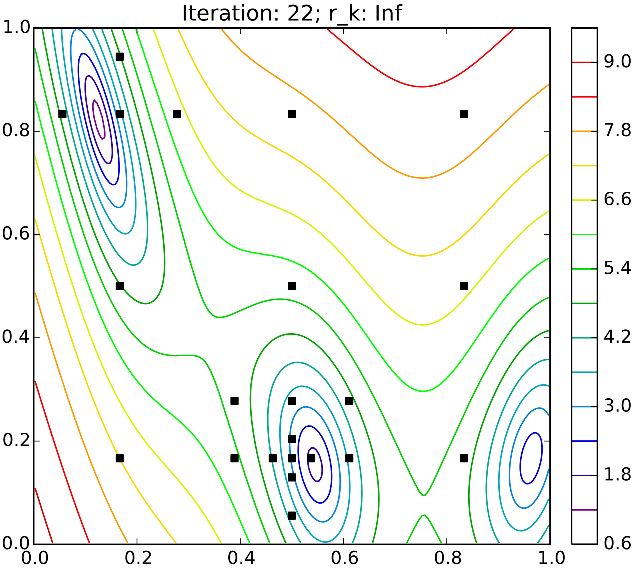
Iteration: 20; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



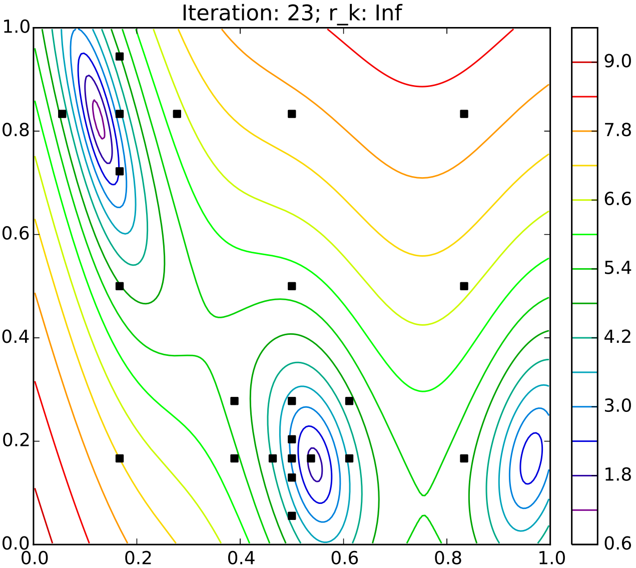
Iteration: 21; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT

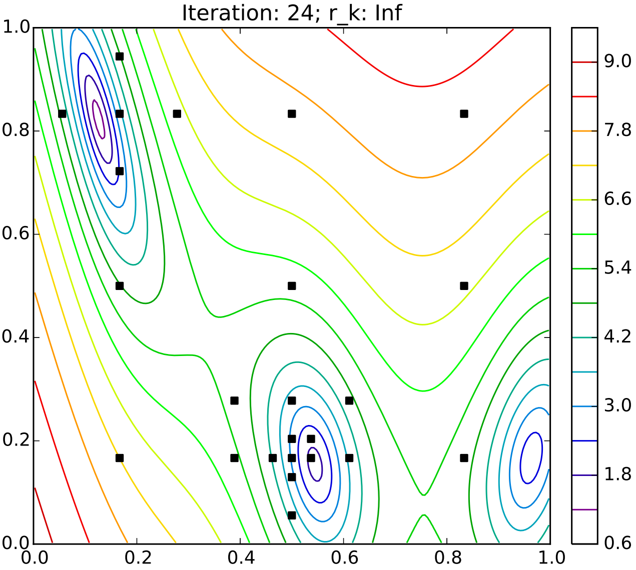


Iteration: 22; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT

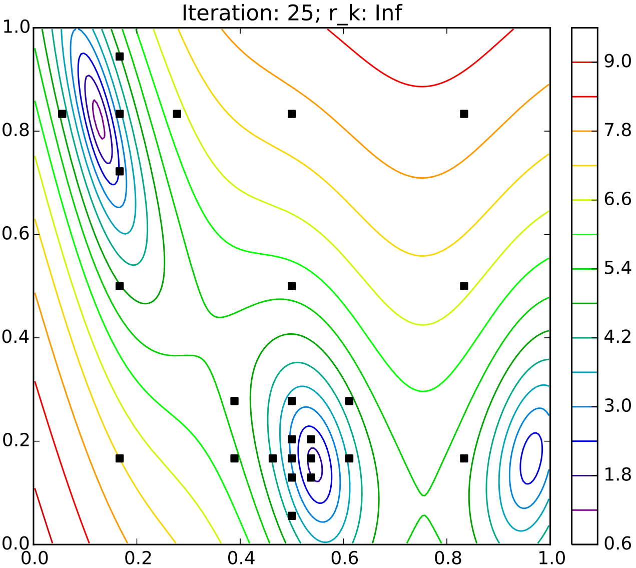

Iteration: 23; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



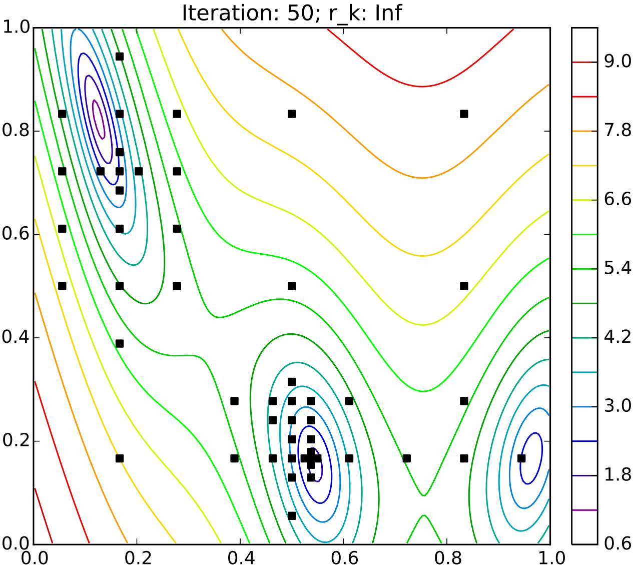
Iteration: 24; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT

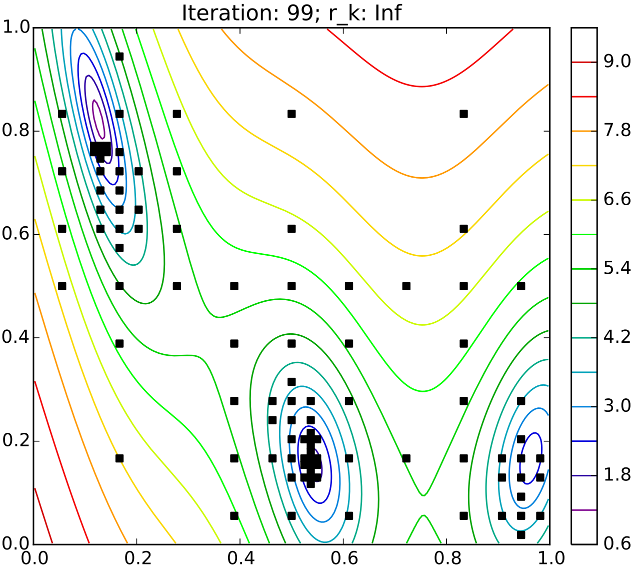


Iteration: 25; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT

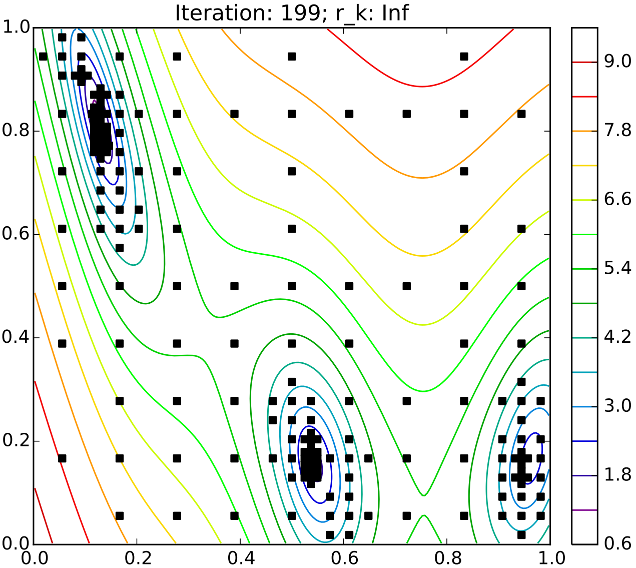

Iteration: 50; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



Iteration: 99; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

DIRECT



Iteration: 199; r_k: Inf
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9.0
7.8
6.6
5.4
4.2
3.0
1.8
0.6

Multistart Methods

Given some local optimization routine L:

Algorithm 1: General Multistart

for $k = 1, 2, \ldots$ **do**

  Evaluate f at N points drawn from $\mathcal{D}$

  Start L at some set (possibly empty) of previously evaluated points

Multistart Methods

Given some local optimization routine L:

Algorithm 1: General Multistart

for $k = 1, 2, \ldots$ **do**
  Evaluate f at N points drawn from $\mathcal{D}$
  Start L at some set (possibly empty) of previously evaluated points

- Get to use problem specific local optimization routines.
- Possibly multiple levels of parallelism (objective, local method, global method); L may involve many sequential evaluations of f...

Multistart Methods

Given some local optimization routine L:

Algorithm 1: General Multistart

for $k = 1, 2, \ldots$ **do**

  Evaluate f at N points drawn from $\mathcal{D}$

  Start L at some set (possibly empty) of previously evaluated points

- Get to use problem specific local optimization routines.
- Possibly multiple levels of parallelism (objective, local method, global method); L may involve many sequential evaluations of f...
- Which points should start runs?
- If resources are limited, how should points from each run receive priority?

Multistart Methods

Given some local optimization routine L:

Algorithm 1: General Multistart

for $k = 1, 2, \dots$ **do**
- Evaluate f at N points drawn from $\mathcal{D}$
- Start L at some set (possibly empty) of previously evaluated points

- Get to use problem specific local optimization routines.
- Possibly multiple levels of parallelism (objective, local method, global method); L may involve many sequential evaluations of f...

- Which points should start runs?
- If resources are limited, how should points from each run receive priority?

- Ideally, only one run is started for each minima.
- Exploring by sampling. Refining with L.

Multi-Level Single Linkage

Given some local optimization routine L:

Algorithm 2: MLSL

for $k = 1, 2, \dots$ **do**

Sample f at N random points drawn uniformly from $\mathcal{D}$

Start L at all sample points x:

- that has yet to start a run
- $\nexists x_i : \|x - x_i\| \leq r_k$ and $f(x_i) < f(x)$

[Rinnooy Kan and Timmer, *Mathematical Programming*, 39(1):57–78, 1987]

Multi-Level Single Linkage

Given some local optimization routine L:

Algorithm 2: MLSL

for $k = 1, 2, \ldots$ **do**

Sample f at N random points drawn uniformly from $\mathcal{D}$

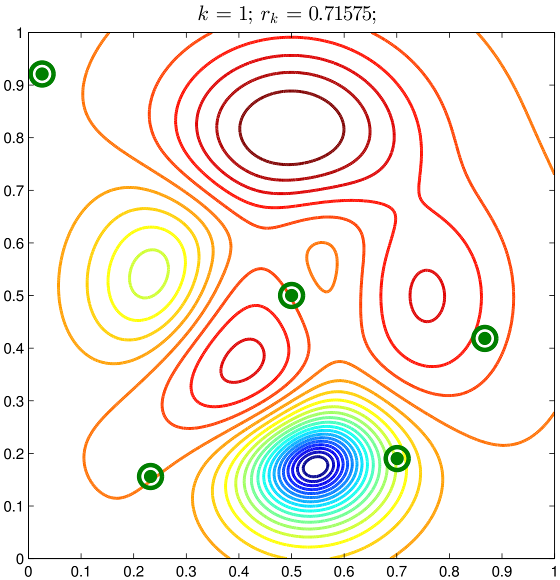
Start L at all sample points x:

- that has yet to start a run
- $\nexists x_i : \|x - x_i\| \leq r_k$ and $f(x_i) < f(x)$

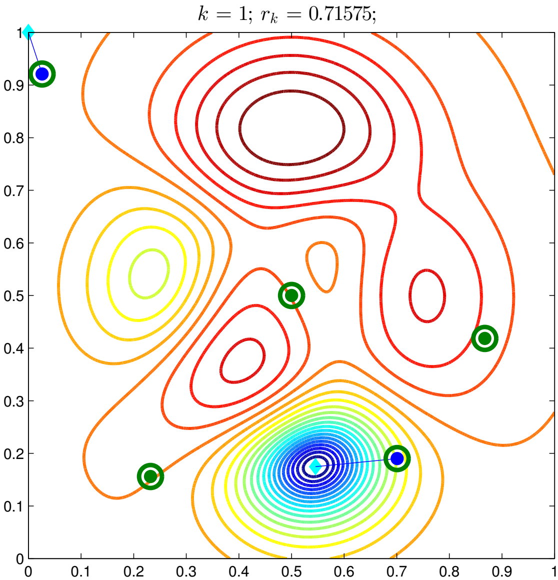
[Rinnooy Kan and Timmer, *Mathematical Programming*, 39(1):57–78, 1987]

- Doesn't naturally translate when evaluations of f are limited
- Ignores some points when deciding where to start L

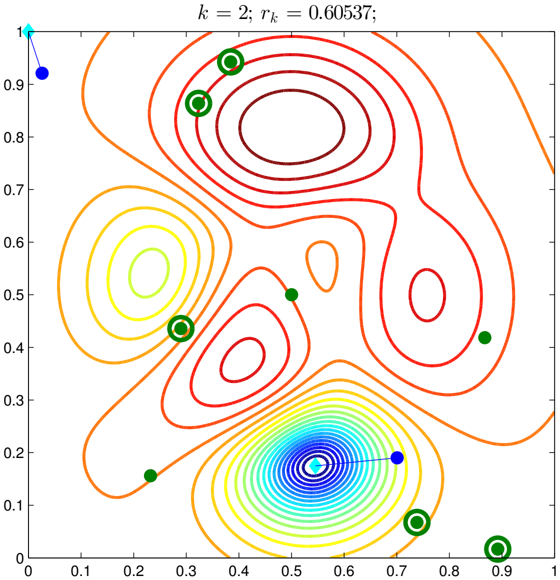
Multi-Level Single Linkage



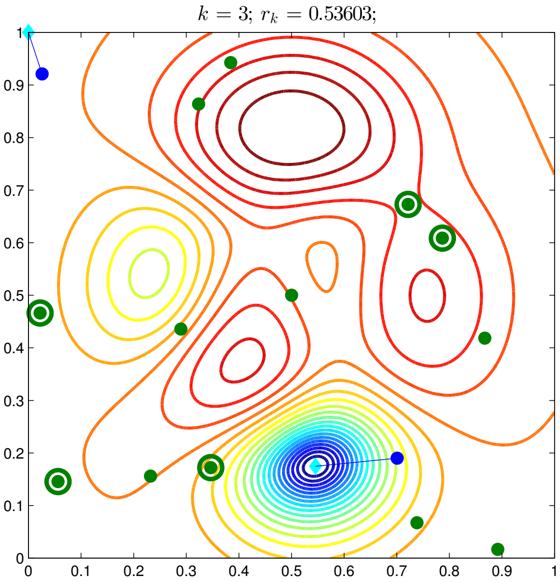
Multi-Level Single Linkage



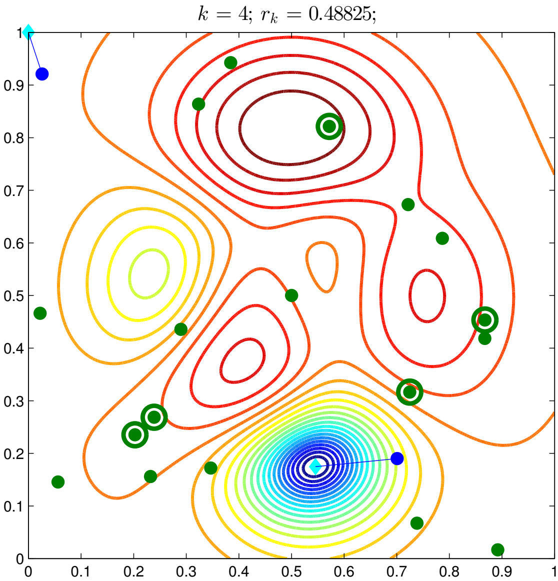
Multi-Level Single Linkage



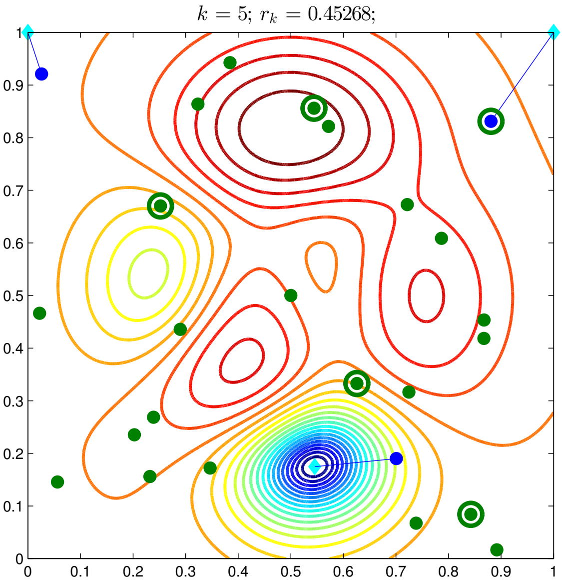
Multi-Level Single Linkage



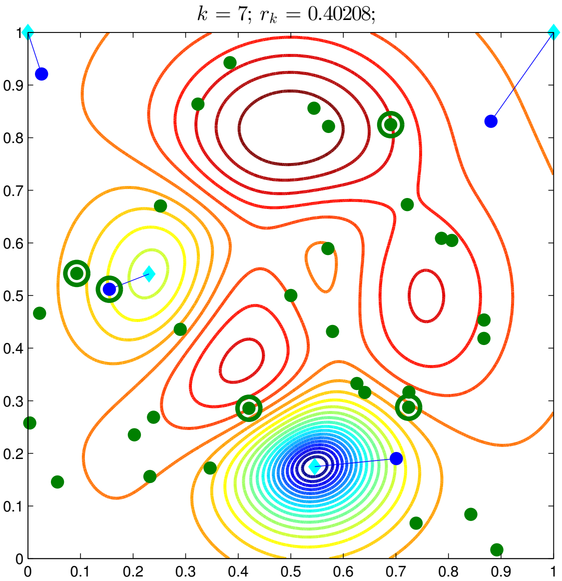
Multi-Level Single Linkage



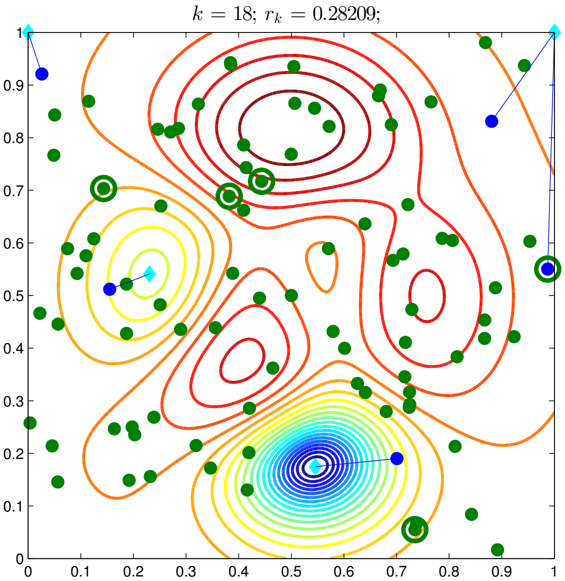
Multi-Level Single Linkage



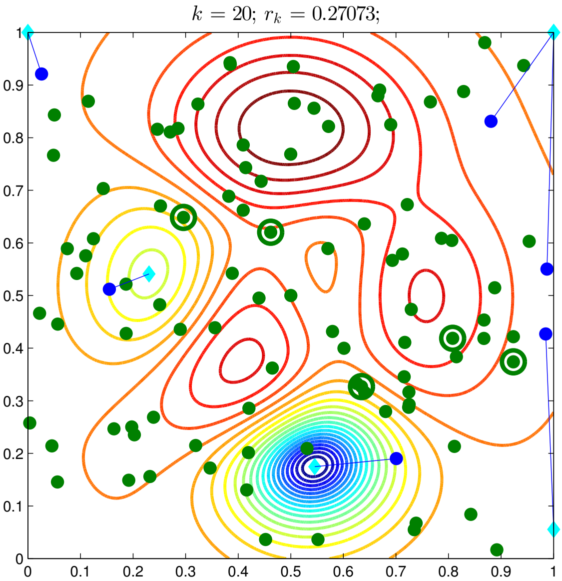
Multi-Level Single Linkage



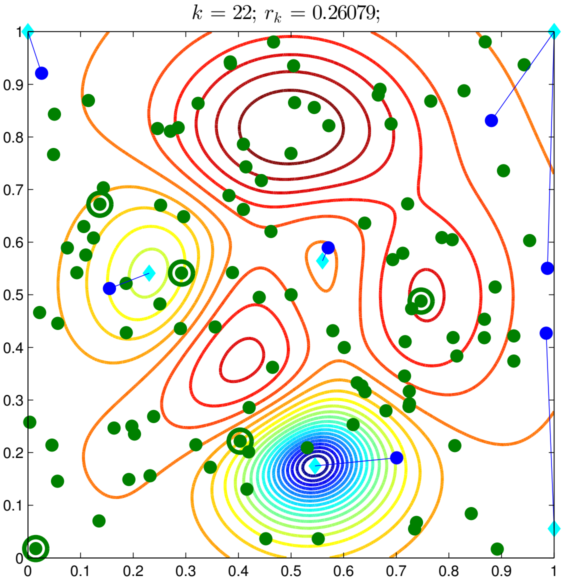
Multi-Level Single Linkage



Multi-Level Single Linkage



Multi-Level Single Linkage



Multi-Level Single Linkage

- $f \in C^1$, with local minima in the interior of $\mathcal{D}$, and the distance between these minima is bounded away from zero.
- L is strictly descent and converges to a minimum (not a stationary point).
- $$r_k = \frac{1}{\sqrt{\pi}} \sqrt[n]{\Gamma\left(1 + \frac{n}{2}\right) \mathrm{vol}\left(\mathcal{D}\right) \frac{\sigma \log kN}{kN}} \qquad (1)$$

Multi-Level Single Linkage

- $f \in C^1$, with local minima in the interior of $\mathcal{D}$, and the distance between these minima is bounded away from zero.
- L is strictly descent and converges to a minimum (not a stationary point).
- $$r_k = \frac{1}{\sqrt{\pi}} \sqrt[n]{\Gamma\left(1 + \frac{n}{2}\right) \mathrm{vol}\left(\mathcal{D}\right) \frac{\sigma \log kN}{kN}} \tag{1}$$

Theorem

If $r_k \to 0$, all local minima will be found almost surely.

Multi-Level Single Linkage

- $f \in C^1$, with local minima in the interior of $\mathcal{D}$, and the distance between these minima is bounded away from zero.

- L is strictly descent and converges to a minimum (not a stationary point).

- $$r_k = \frac{1}{\sqrt{\pi}} \sqrt[n]{\Gamma\left(1 + \frac{n}{2}\right) \mathrm{vol}\left(\mathcal{D}\right) \frac{\sigma \log kN}{kN}} \tag{1}$$

Theorem

If $r_k \to 0$, all local minima will be found almost surely.

If r_k is defined by (1) with $\sigma > 4$, even if the sampling continues forever, the total number of local searches started is finite almost surely.

POSMM

MLSL: (S2)–(S4)

$\hat{x} \in S_k$

(S2) $\nexists x \in S_k$ *with*
$[\|\hat{x} - x\| \leq r_k$ *and* $f(x) < f(\hat{x})]$

(S3) $\hat{x}$ *has not started a local optimization run*

(S4) $\hat{x}$ *is at least* μ *from* $\partial\mathcal{D}$ *and* ν *from known local minima*

POSMM

MLSL: (S2)–(S4)

$\hat{x} \in S_k$

(S1) $\nexists x \in L_k$ with $[\|\hat{x} - x\| \leq r_k$ and $f(x) < f(\hat{x})]$

(S2) $\nexists x \in S_k$ *with* $[\|\hat{x} - x\| \leq r_k$ *and* $f(x) < f(\hat{x})]$

(S3) $\hat{x}$ *has not started a local optimization run*

(S4) $\hat{x}$ *is at least* μ *from* $\partial\mathcal{D}$ *and* ν *from known local minima*

POSMM: (S1)–(S4), (L1)–(L6)

$\hat{x} \in L_k$

(L1) $\nexists x \in L_k$ $[\|\hat{x} - x\| \leq r_k$ and $f(x) < f(\hat{x})]$

(L2) $\nexists x \in S_k$ with $[\|\hat{x} - x\| \leq r_k$ and $f(x) < f(\hat{x})]$

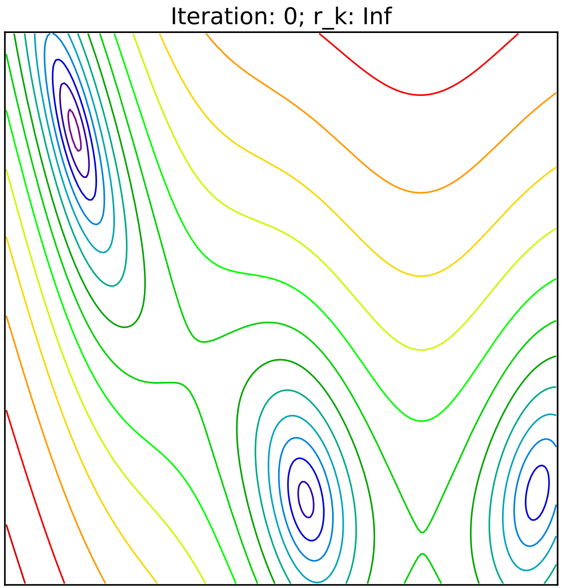
(L3) $\hat{x}$ has not started a local optimization run

(L4) $\hat{x}$ is at least μ from $\partial\mathcal{D}$ and ν from known local minima

(L5) $\hat{x}$ is not in an active local optimization run and has not been ruled stationary

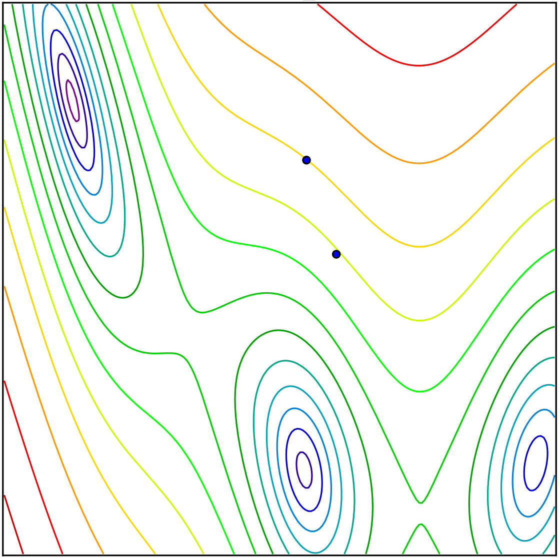
(L6) $\exists r_k$-descent path in H_k from some $x \in S_k$ satisfying (S2-S4) to $\hat{x}$

POSMM

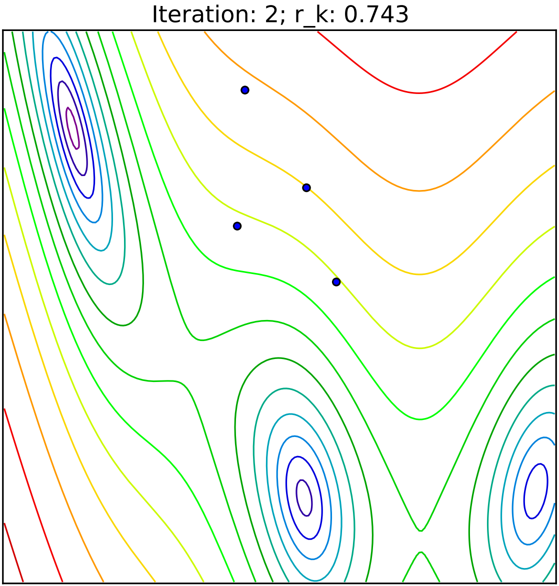


POSMM

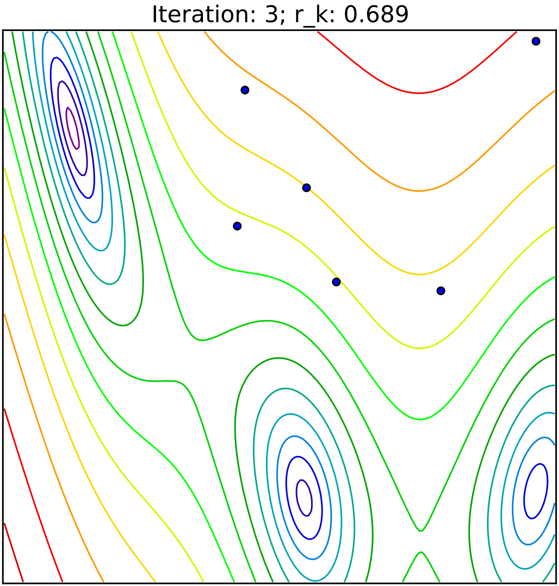
Iteration: 1; r_k: 0.743



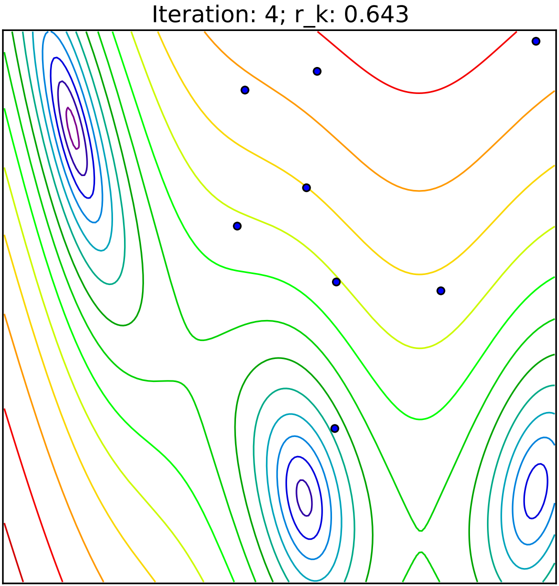
POSMM



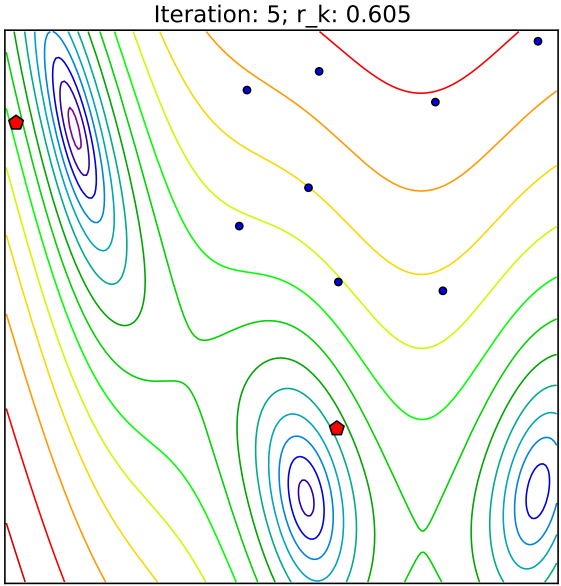
POSMM



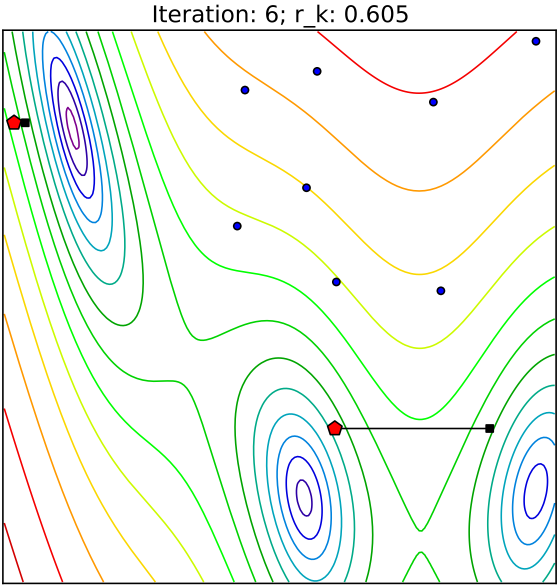
POSMM



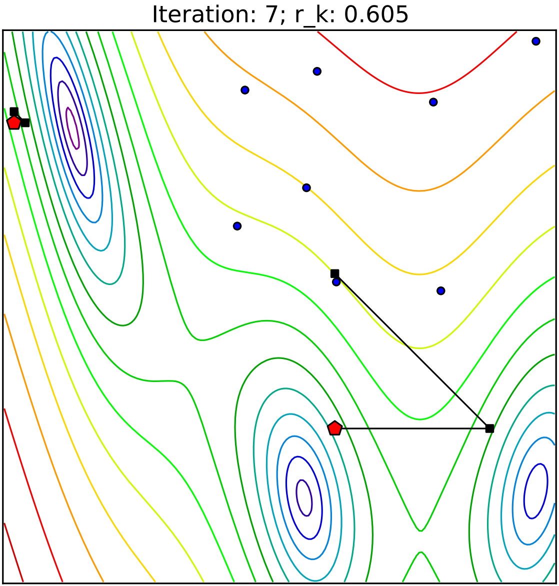
POSMM



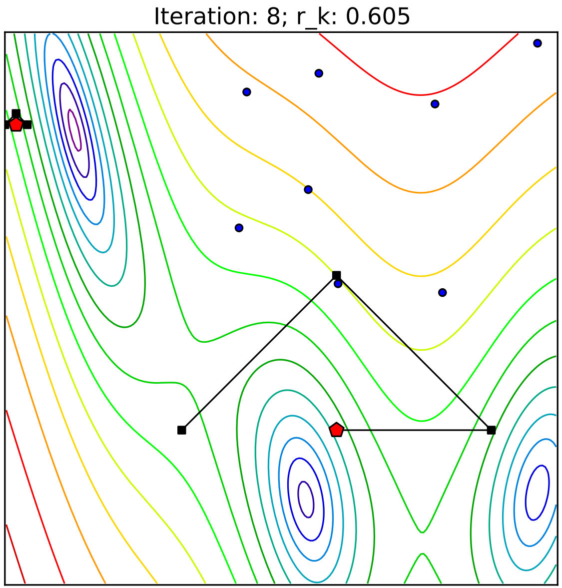
POSMM



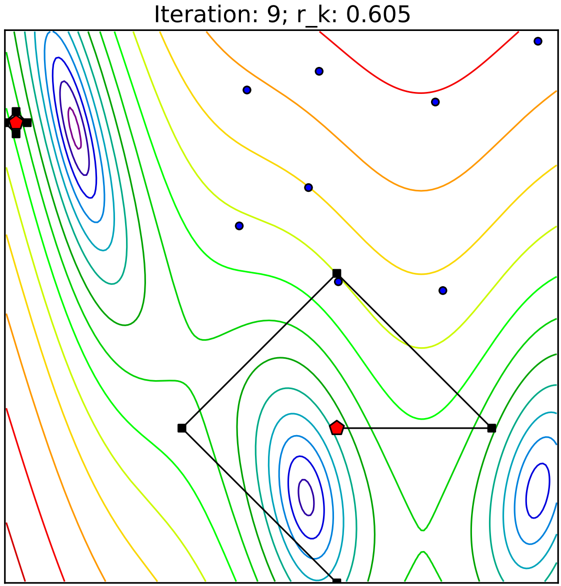
POSMM



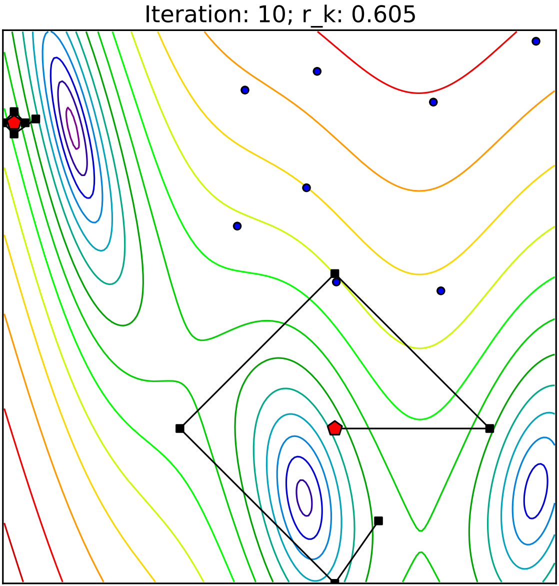
POSMM



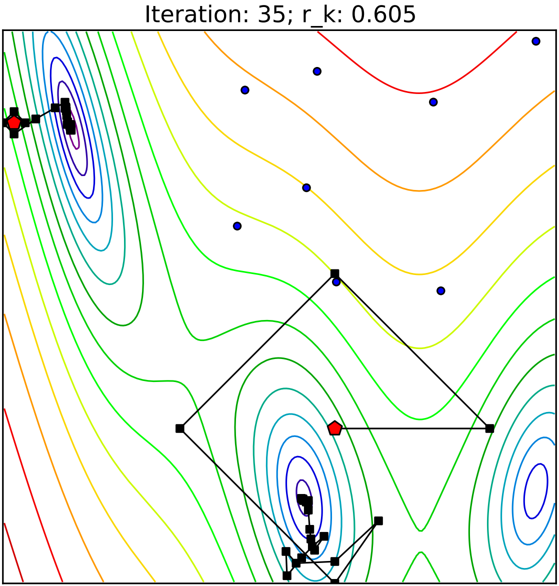
POSMM



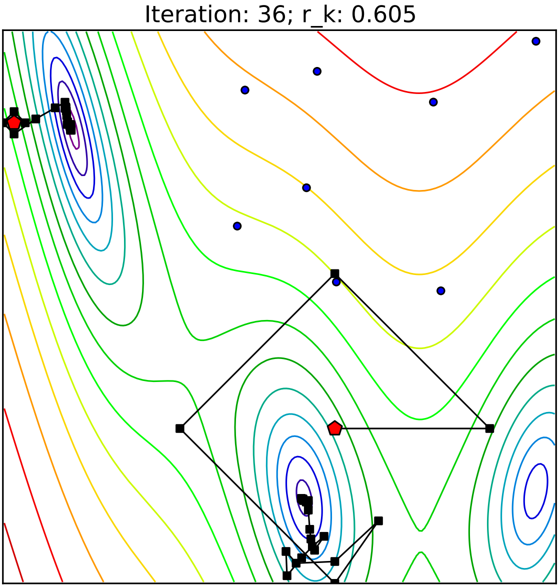
POSMM



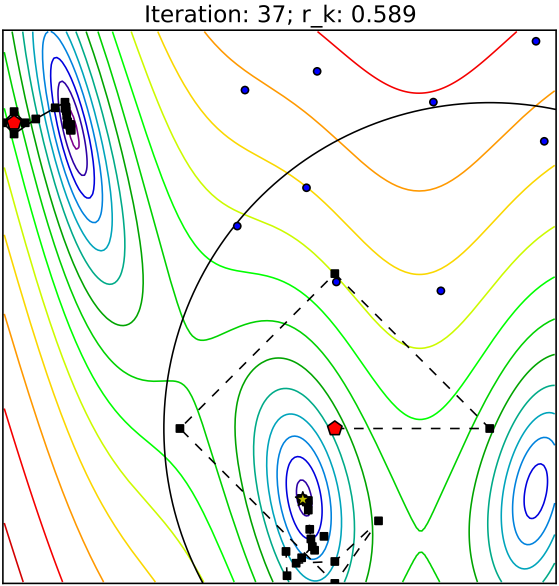
POSMM



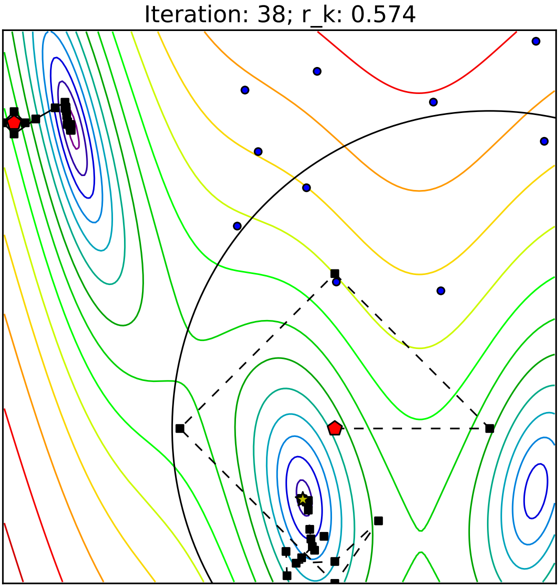
POSMM



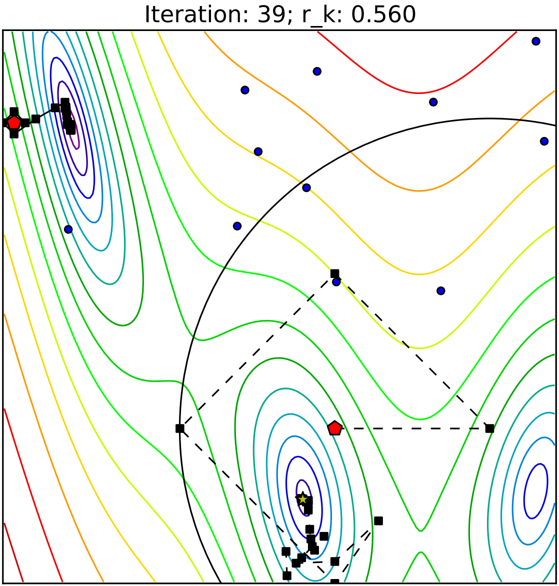
POSMM



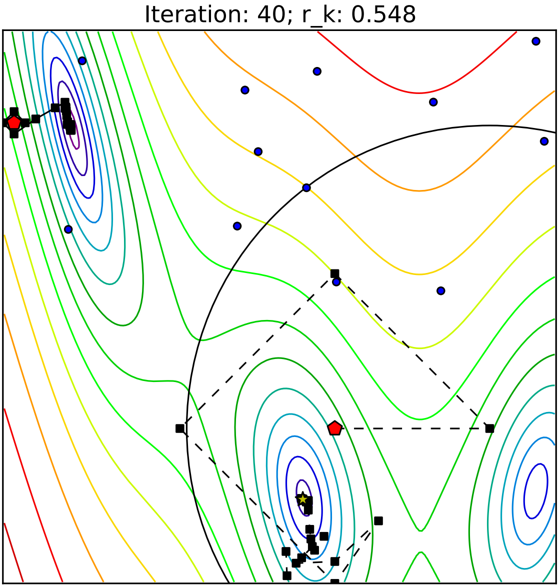
POSMM



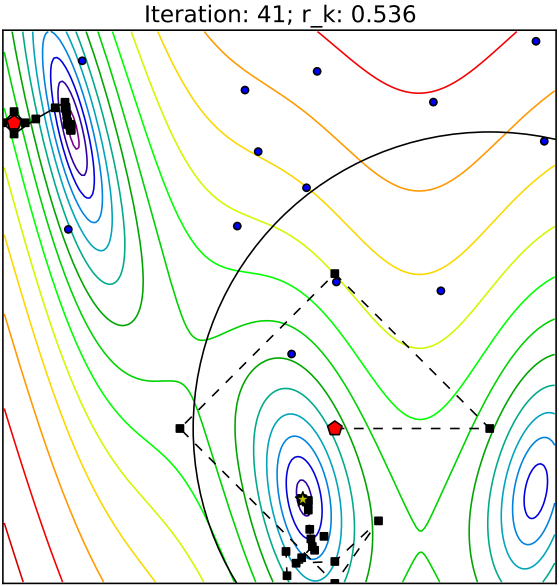
POSMM



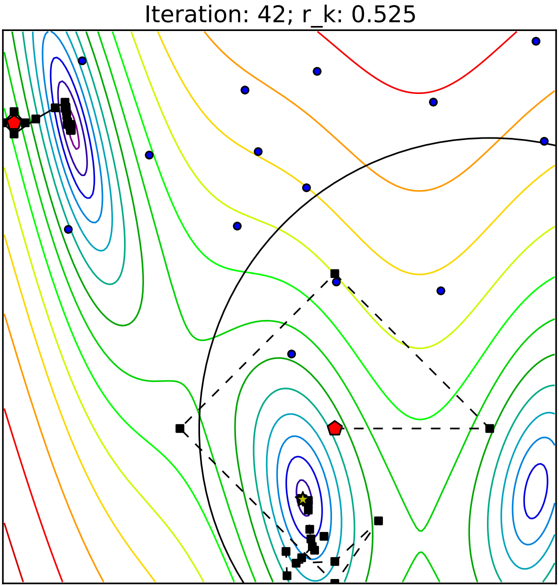
POSMM



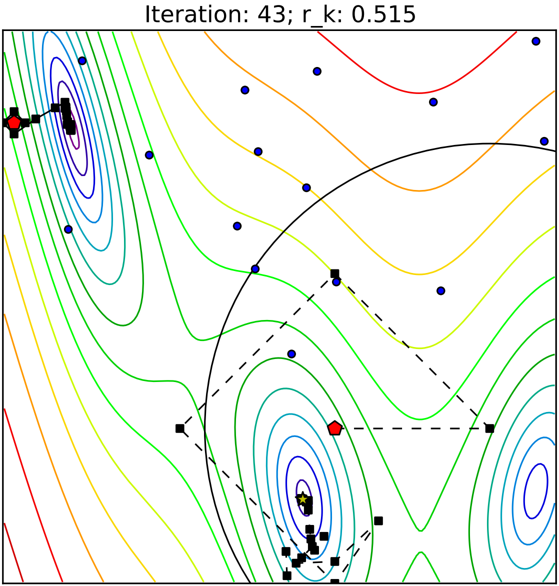
POSMM



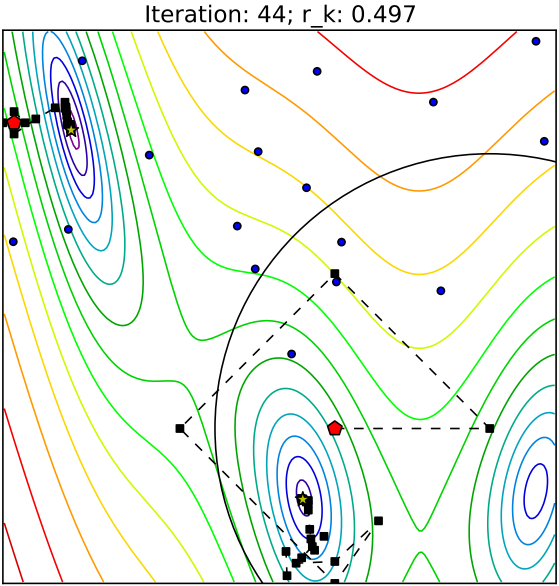
POSMM



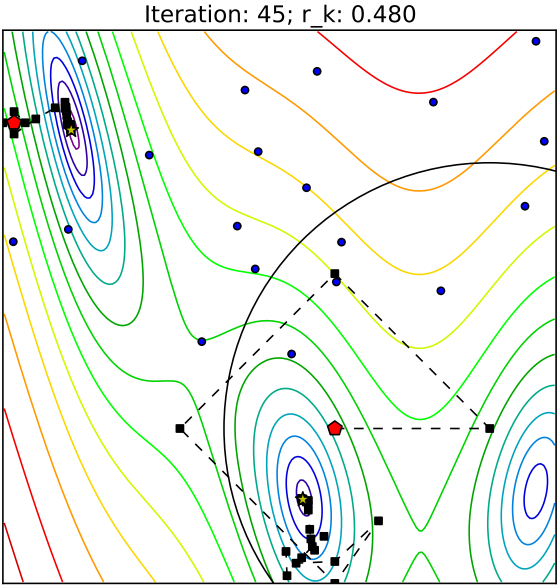
POSMM



POSMM

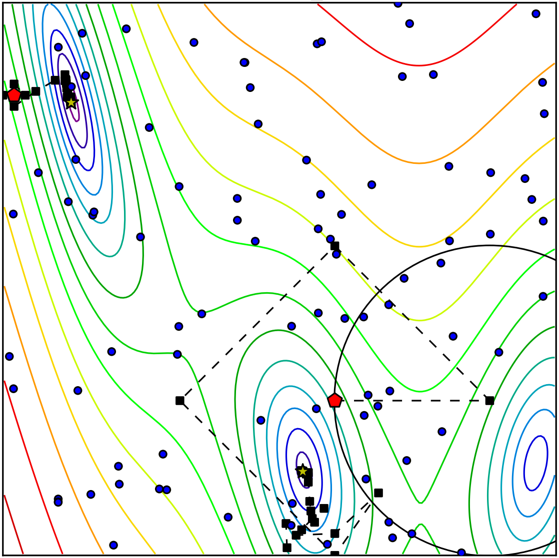


POSMM



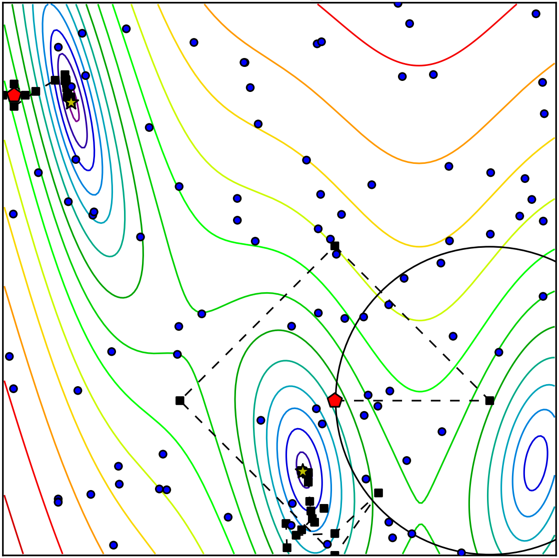
POSMM

Iteration: 80; r_k: 0.281

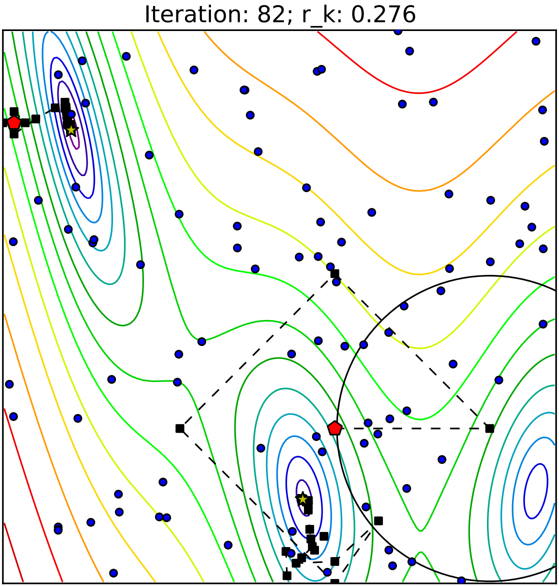


POSMM

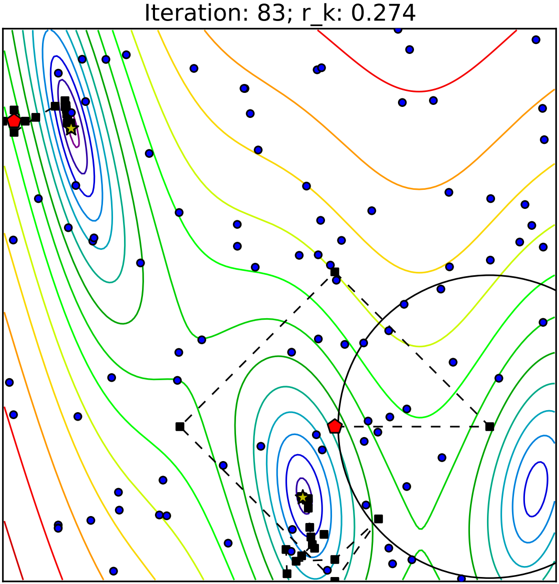
Iteration: 81; r_k: 0.279



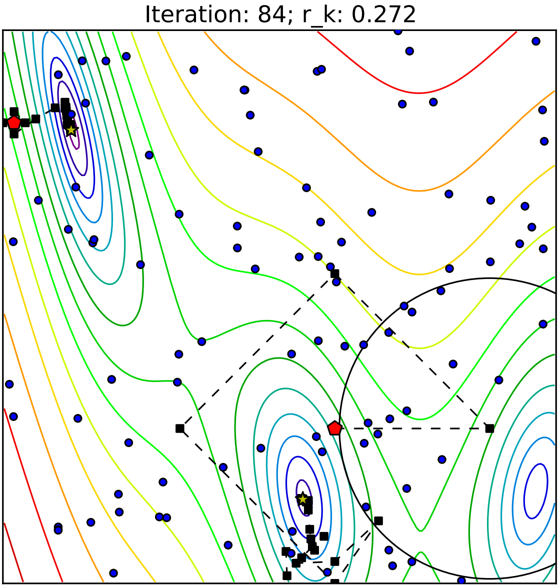
POSMM



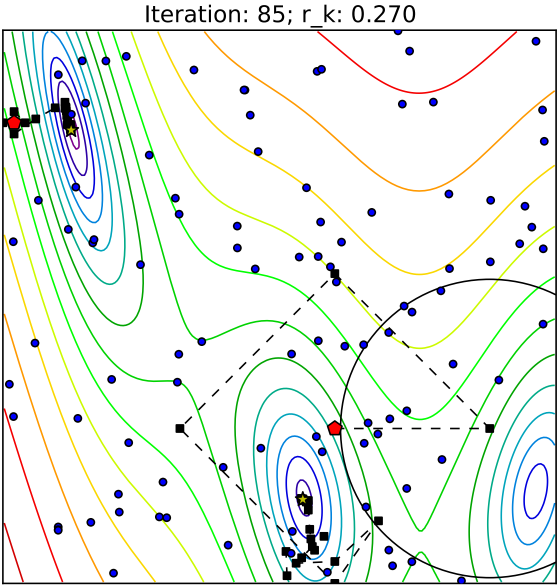
POSMM



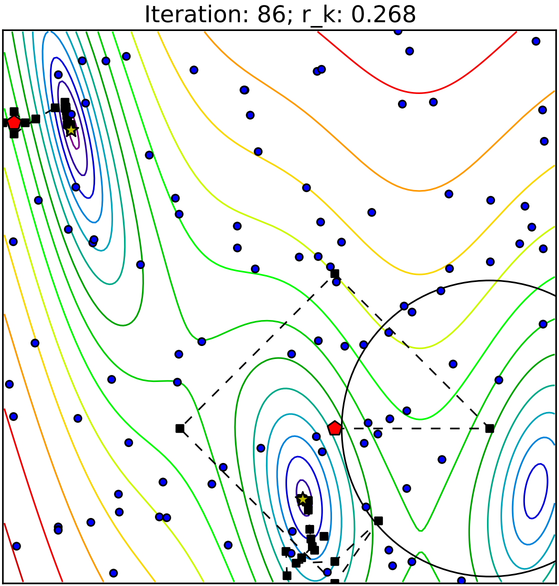
POSMM



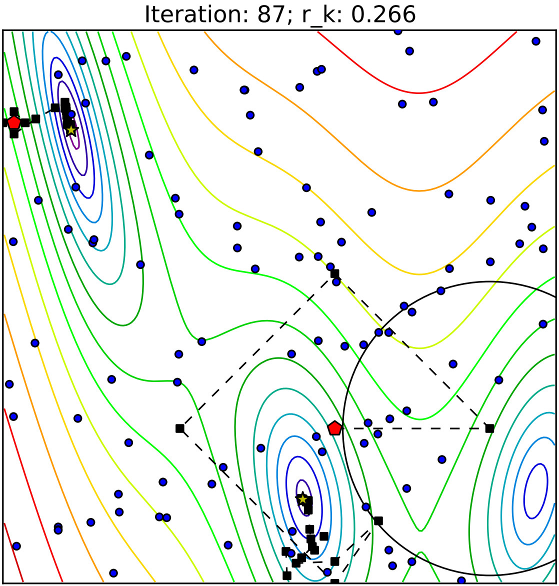
POSMM



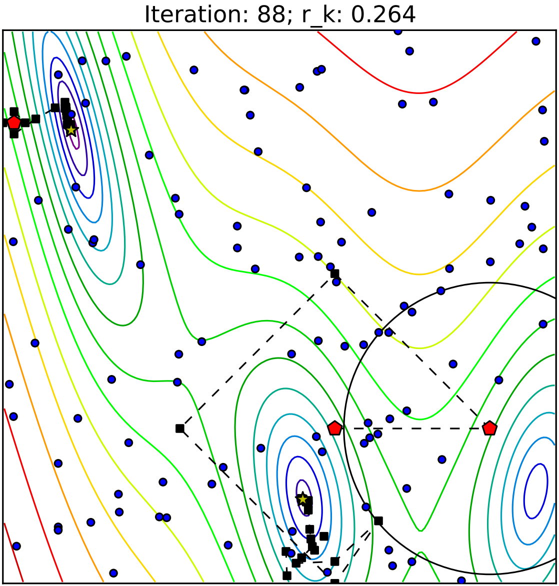
POSMM



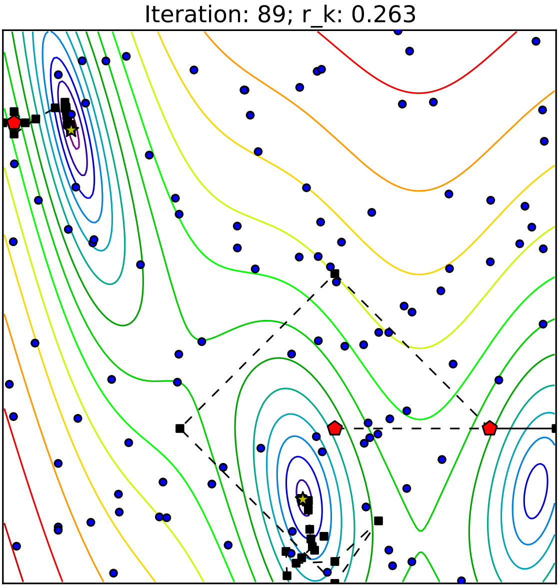
POSMM



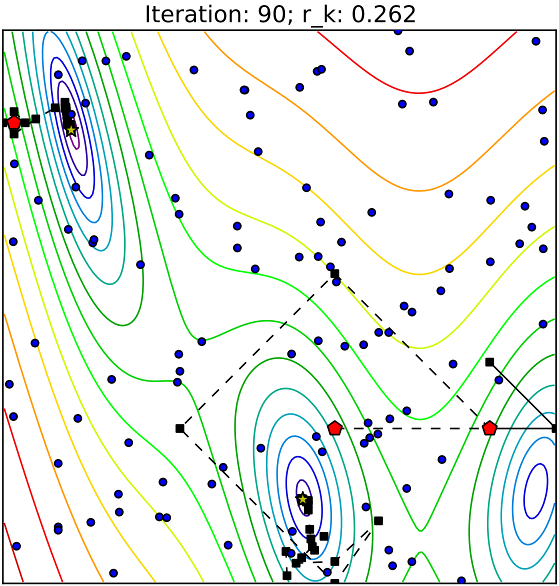
POSMM



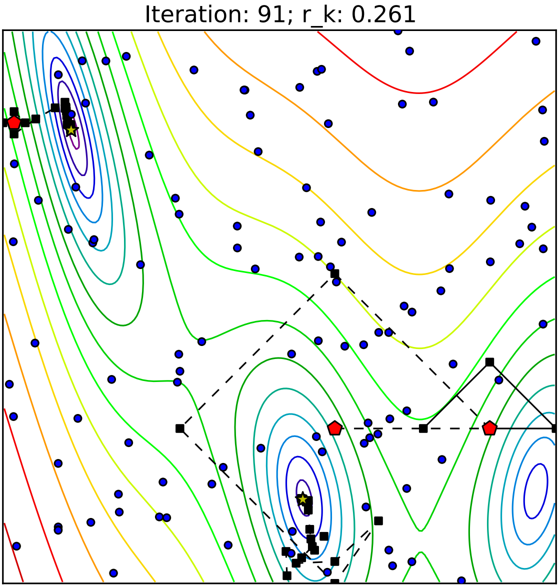
POSMM



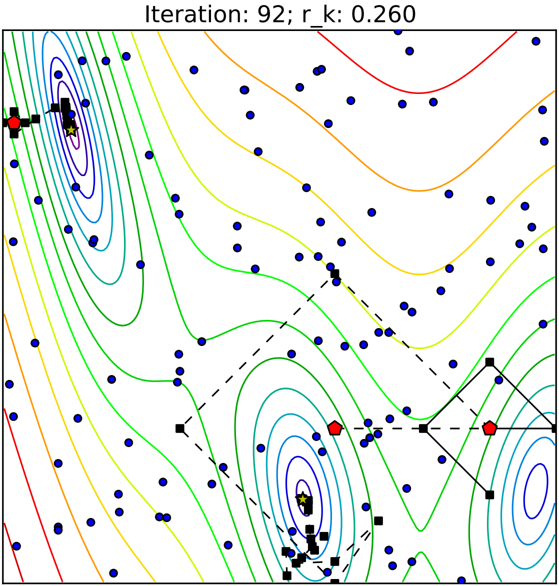
POSMM



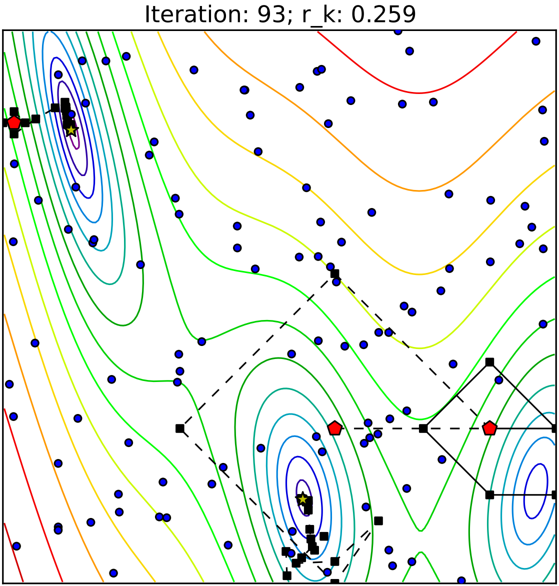
POSMM



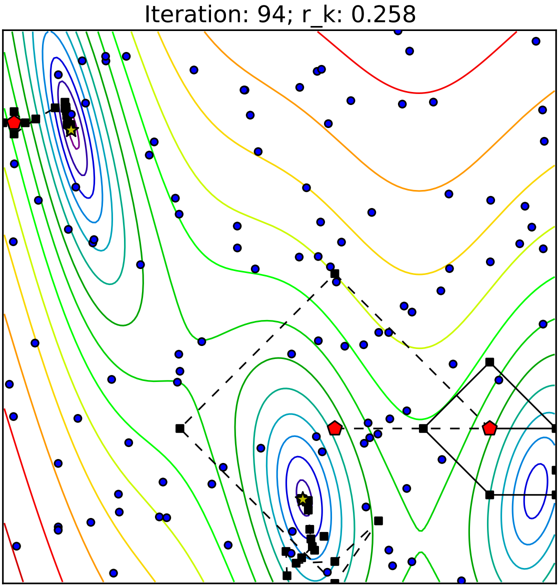
POSMM



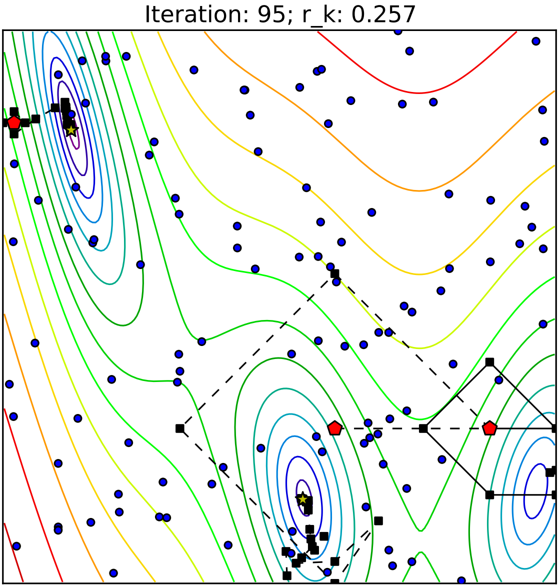
POSMM



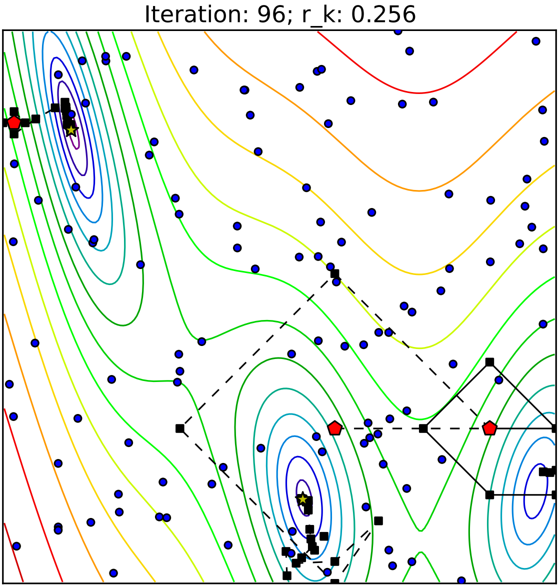
POSMM



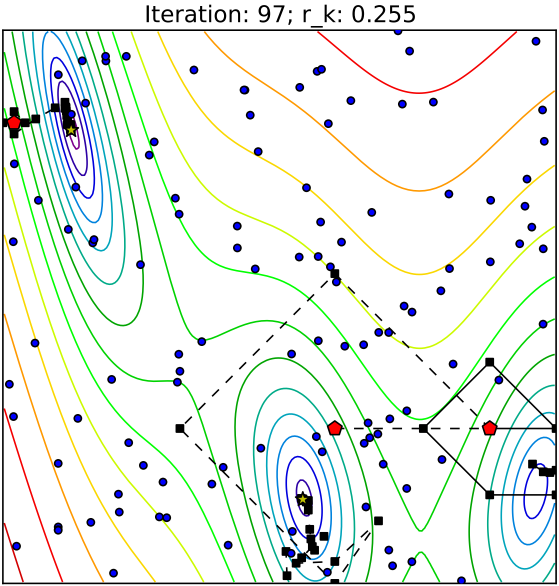
POSMM



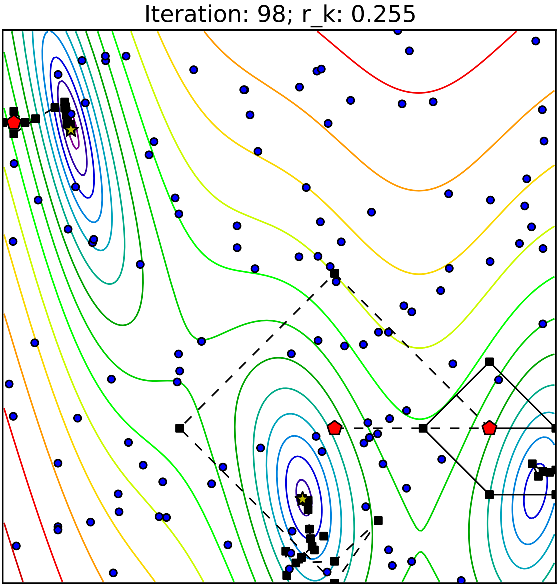
POSMM



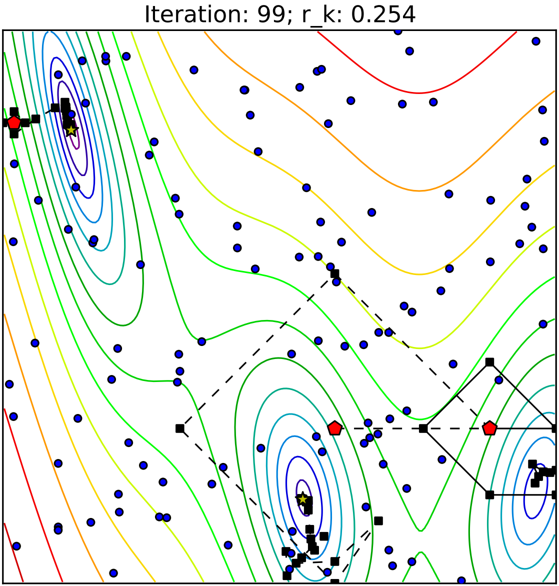
POSMM



POSMM



POSMM



Properties of the local optimization method

Necessary:

- Honors a starting point
- Honors bound constraints

Properties of the local optimization method

Necessary:

- Honors a starting point
- Honors bound constraints

ORBIT satisfies these [Wild, Regis, Shoemaker, SIAM-JOSC, 2008]

BOBYQA satisfies these [Powell, 2009]

Properties of the local optimization method

Necessary:

- Honors a starting point
- Honors bound constraints

ORBIT satisfies these [Wild, Regis, Shoemaker, SIAM-JOSC, 2008]

BOBYQA satisfies these [Powell, 2009]

Possibly beneficial:

- Can return multiple points of interest
- Reports solution quality/confidence at every iteration
- Can avoid certain regions in the domain
- Uses a history of past evaluations of f
- Uses additional points mid-run

APOSSM Theory

Theorem

Given the same assumptions as MLSL, APOSSM will start a finite number of local optimization runs with probability 1.

APOSSM Theory

Theorem

Given the same assumptions as MLSL, APOSSM will start a finite number of local optimization runs with probability 1.

Assumption

There exists $K_0 < \infty$ so that for any K_0 consecutive iterations, there is a positive (bounded away from zero) probability of evaluating a point from the sample stream and each existing local optimization run.

APOSSM Theory

Theorem

Given the same assumptions as MLSL, APOSSM will start a finite number of local optimization runs with probability 1.

Assumption

There exists $K_0 < \infty$ so that for any K_0 consecutive iterations, there is a positive (bounded away from zero) probability of evaluating a point from the sample stream and each existing local optimization run.

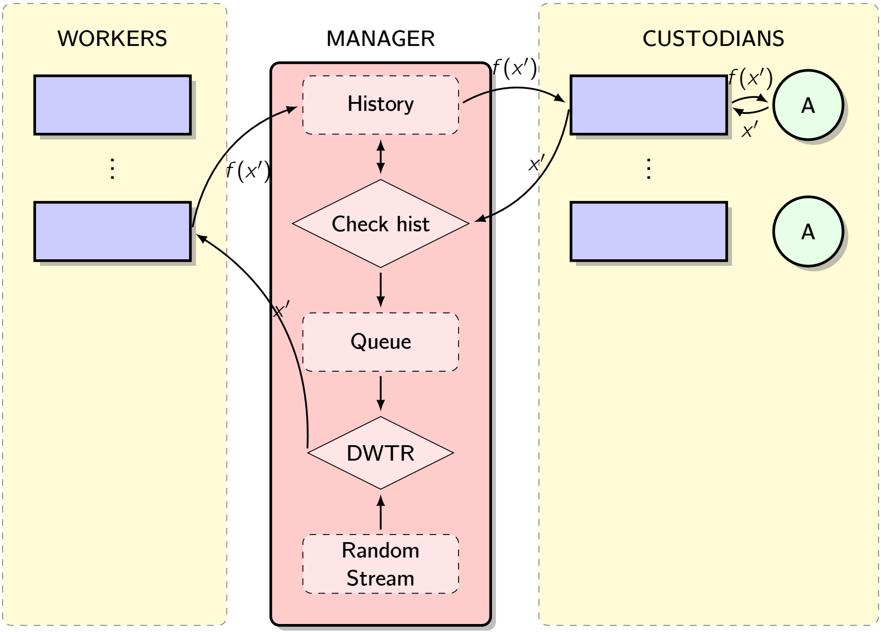
Theorem

Each $x^* \in X^*$ will almost surely be either identified in a finite number of evaluations or have a single local optimization run that is converging asymptotically to it.

Measuring Performance

- **GLODS** Global & local optimization using direct search [Custódio, Madeira (JOGO, 2014)]
- **Direct** Serial DIRECT [D. Finkel's MATLAB code]
- **pVTDirect** Parallel DIRECT [He, Watson, Sosonkina (TOMS, 2009)]
- **Random** Uniform sampling over domain (as a baseline)
- **POSMM**
  - Local optimization method
    - ORBIT [Wild, Regis, & Shoemaker (SIAM JOSC, 2008)]
    - BOBYQA [Powell, 2009]
  - Initial sample size: $10n$

- Each method evaluates Direct's $2n + 1$ initial points.



(A)POSMM Diagram

(A)POSMM Manager

Algorithm 3: Manager Logic

while $k < \text{fevalmax}$ **do**
    MPI.recv(MPI.ANY_SOURCE)
    if *Received from Custodian* **then**
        Check H_k
        Add new point to Q_L
    if *Received from Worker* **then**
        Update H_k
        Possibly get a Custodian working on the next point
        Run `decide_where_to_start`
        Possibly update Q_L
    if sync=False *OR All Workers/Custodians are done* **then**
        Give from Q_L or $\mathcal{R}_S$ to available worker(s)

POSMM

MLSL: (S2)–(S4)

$\hat{x} \in S_k$

(S1) $\nexists x \in L_k$ with $[\|\hat{x} - x\| \leq r_k$ and $f(x) < f(\hat{x})]$

(S2) $\nexists x \in S_k$ *with* $[\|\hat{x} - x\| \leq r_k$ *and* $f(x) < f(\hat{x})]$

(S3) $\hat{x}$ *has not started a local optimization run*

(S4) $\hat{x}$ *is at least* μ *from* $\partial\mathcal{D}$ *and* ν *from known local minima*

POSMM: (S1)–(S4), (L1)–(L6)

$\hat{x} \in L_k$

(L1) $\nexists x \in L_k$ $[\|\hat{x} - x\| \leq r_k$ and $f(x) < f(\hat{x})]$

(L2) $\nexists x \in S_k$ with $[\|\hat{x} - x\| \leq r_k$ and $f(x) < f(\hat{x})]$

(L3) $\hat{x}$ has not started a local optimization run

(L4) $\hat{x}$ is at least μ from $\partial\mathcal{D}$ and ν from known local minima

(L5) $\hat{x}$ is not in an active local optimization run and has not been ruled stationary

(L6) $\exists r_k$-descent path in H_k from some $x \in S_k$ satisfying (S2-S4) to $\hat{x}$

Measuring Performance

Notation:

Let X^* be the set of all local minima of f.
Let $f^*_{(i)}$ be the ith smallest value $\{f(x^*)|x^* \in X^*\}$.
Let $x^*_{(i)}$ be the element of X^* corresponding to the value $f^*_{(i)}$.

The global minimum has been found at a level $\tau > 0$ at batch k if an algorithm it has found a point $\hat{x}$ satisfying:

$$f(\hat{x}) - f^*_{(1)} \leq (1 - \tau)\left(f(x_0) - f^*_{(1)}\right),$$

where x_0 is the starting point for problem p.

Measuring Performance

Notation:

Let X^* be the set of all local minima of f.
Let $f^*_{(i)}$ be the ith smallest value $\{f(x^*) | x^* \in X^*\}$.
Let $x^*_{(i)}$ be the element of X^* corresponding to the value $f^*_{(i)}$.

The j best local minima have been found at a level $\tau > 0$ at batch k if:

$$\left|\left\{x^*_{(1)}, \ldots, x^*_{(\underline{j}-1)}\right\} \bigcap \left\{x^*_{(i)} : \exists x \in H_k \text{ with } \left\|x - x^*_{(i)}\right\| \le r_n(\tau)\right\}\right| = \underline{j} - 1$$

$$\&$$

$$\left|\left\{x^*_{(\underline{j})}, \ldots, x^*_{(\bar{j})}\right\} \bigcap \left\{x^*_{(i)} : \exists x \in H_k \text{ with } \left\|x - x^*_{(i)}\right\| \le r_n(\tau)\right\}\right| \ge j - \underline{j} + 1,$$

where $\underline{j}$ and $\bar{j}$ are the smallest and largest integers such that $f^*_{(\bar{j})} = f^*_{(j)} = f^*_{(\underline{j})}$ and where $r_n(\tau) = \sqrt[n]{\frac{\tau \operatorname{vol}(\mathcal{D})\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}$.

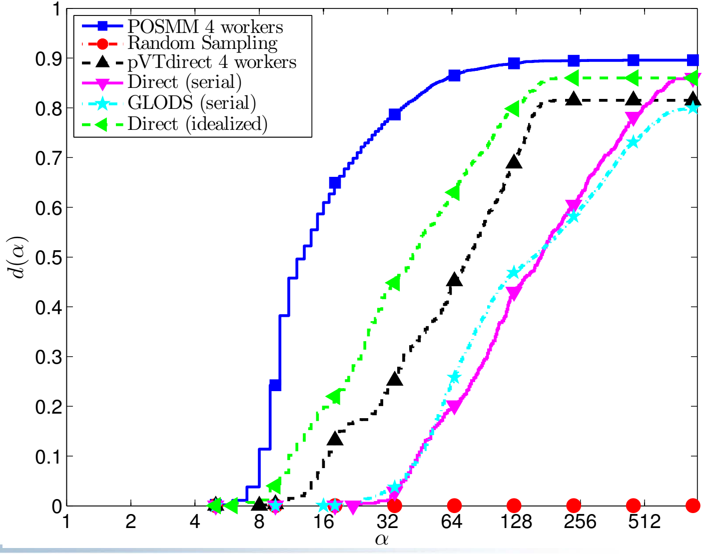
Problems considered

GKLS problem generator [Gaviano et al., "Algorithm 829" (TOMS, 2003)]

- 600 synthetic problems with known local minima
- $n = 2, \dots, 7$
- 10 local minima in the unit cube with a unique global minimum
- 100 problems for each dimension
- 5 replications (different seeds) for each problem
- 5000 evaluations

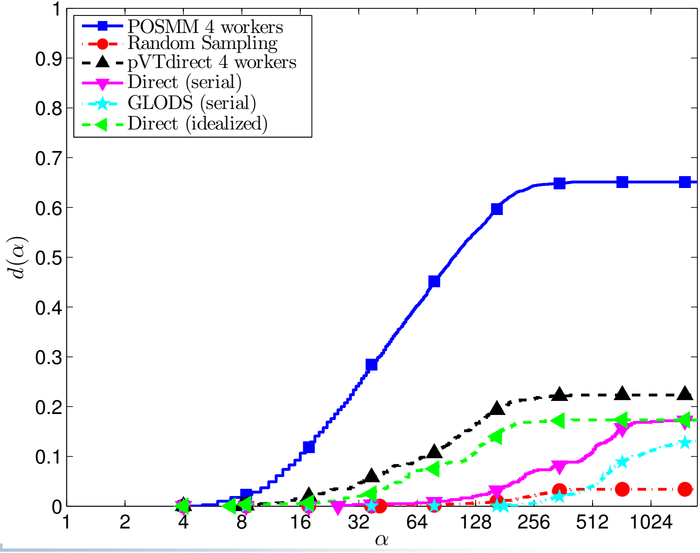
Data Profiles

$$f(x) - f^*_{(1)} \le (1 - 10^{-5}) \left(f(x_0) - f^*_{(1)} \right)$$

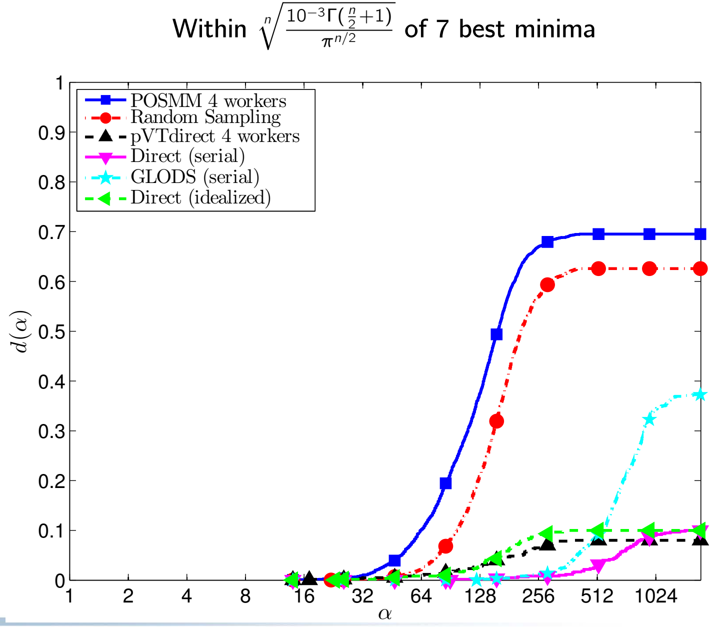


Data Profiles

Within $\sqrt[n]{\frac{10^{-4}\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}$ of 3 best minima

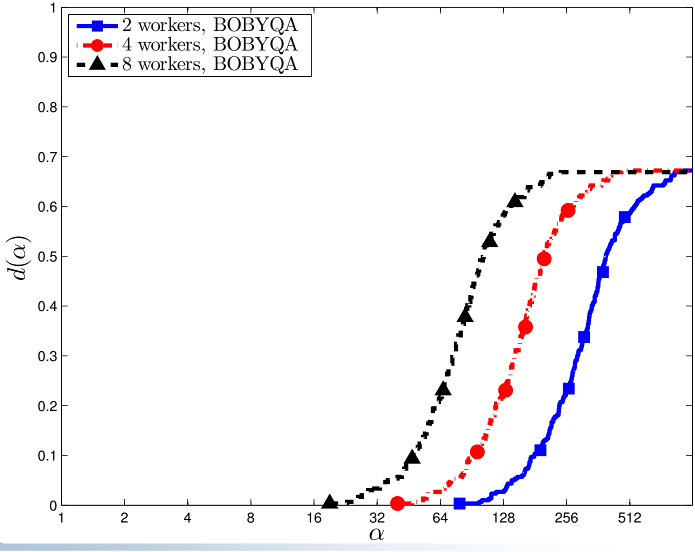


Data Profiles



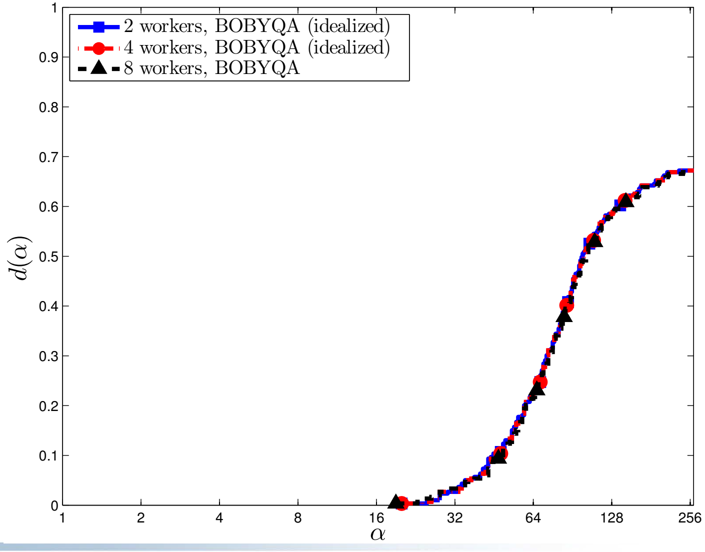
Data Profiles (increasing workers)

Within $\sqrt[n]{\frac{10^{-3}\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}$ of 7 best minima



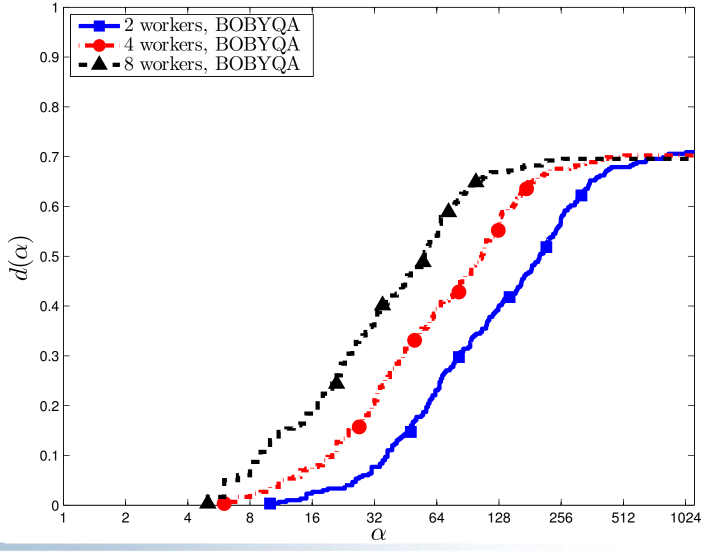
Data Profiles (increasing workers)

Within $\sqrt[n]{\frac{10^{-3}\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}$ of 7 best minima



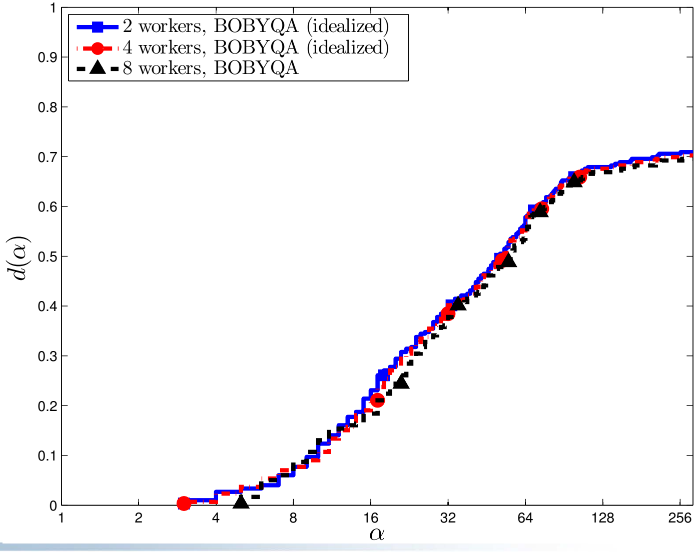
Data Profiles (increasing workers)

Within $\sqrt[n]{\frac{10^{-4}\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}$ of 3 best minima

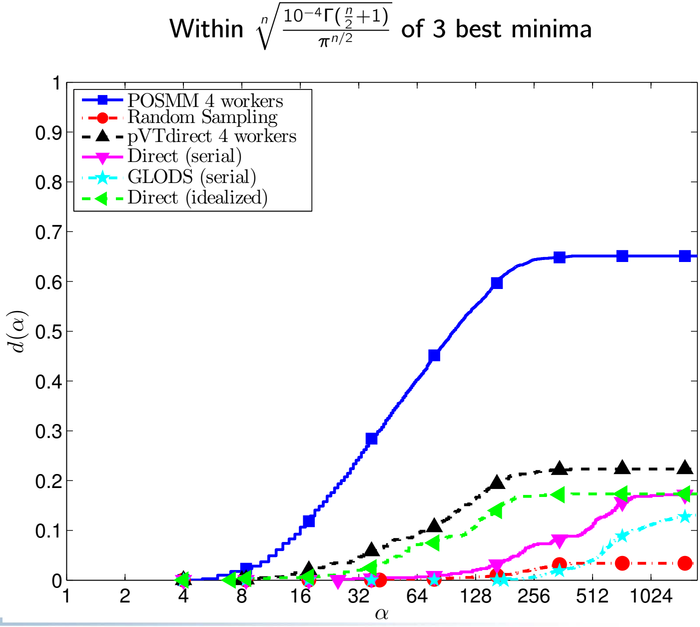


Data Profiles (increasing workers)

Within $\sqrt[n]{\frac{10^{-4}\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}$ of 3 best minima

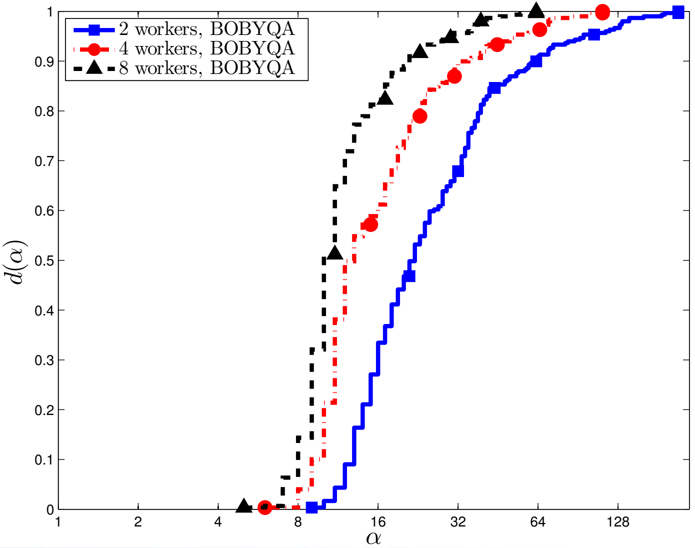


Data Profiles (increasing workers)



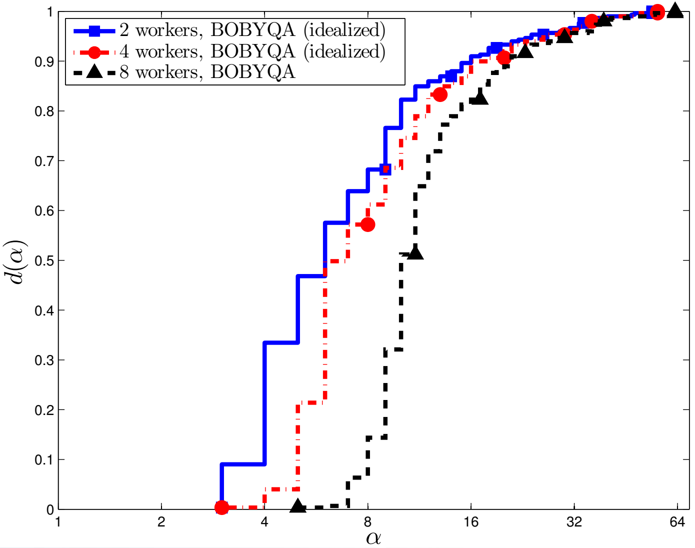
Data Profiles (increasing workers)

$$f(x) - f^*_{(1)} \le (1 - 10^{-5})\left(f(x_0) - f^*_{(1)}\right)$$



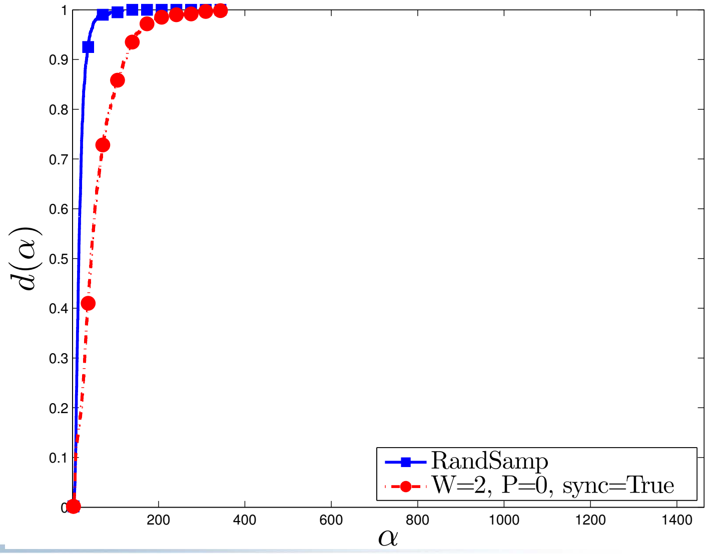
Data Profiles (increasing workers)

$$f(x) - f^*_{(1)} \leq (1 - 10^{-5}) \left(f(x_0) - f^*_{(1)} \right)$$

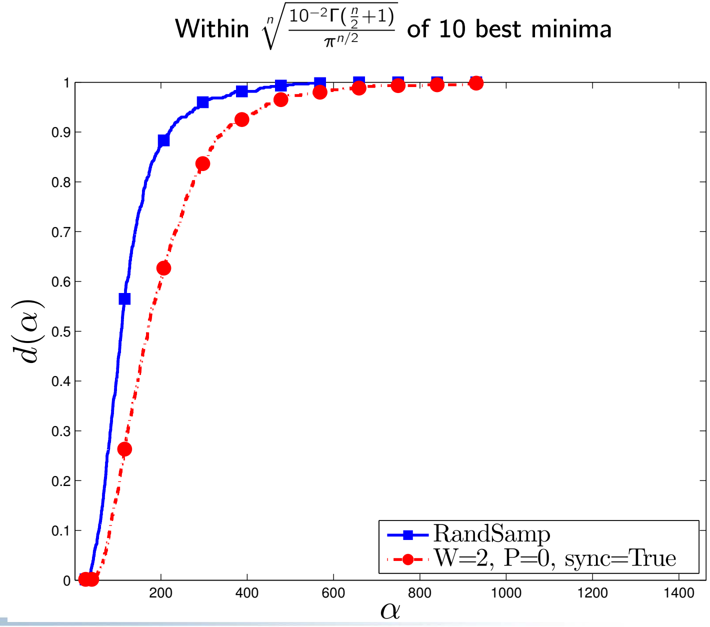


Data Profiles (vs random)

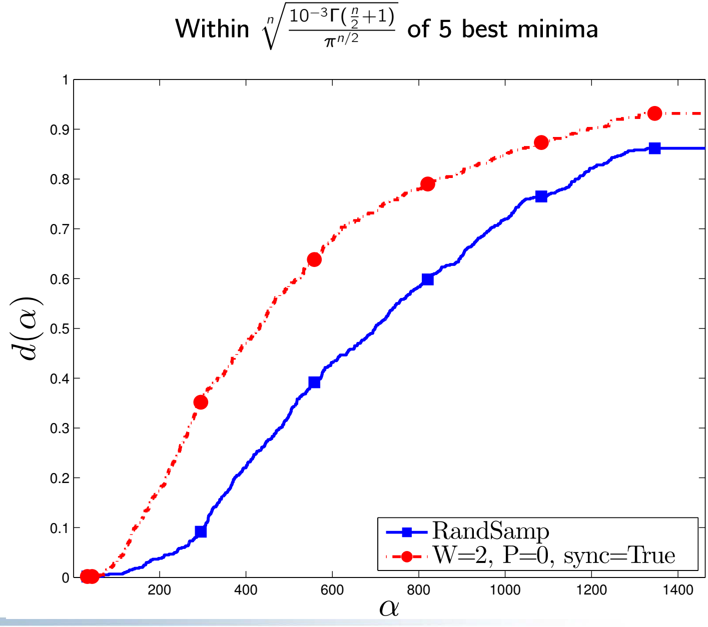
Within $\sqrt[n]{\frac{10^{-1}\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}$ of 10 best minima



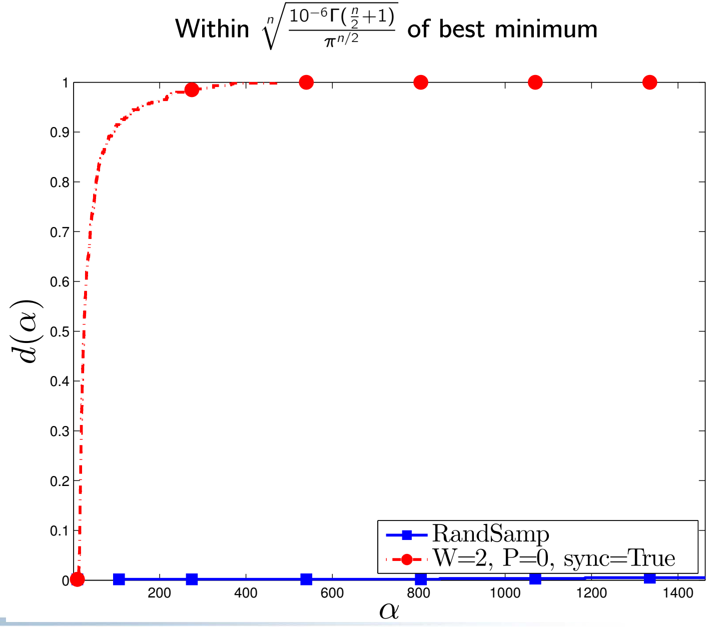
Data Profiles (vs random)



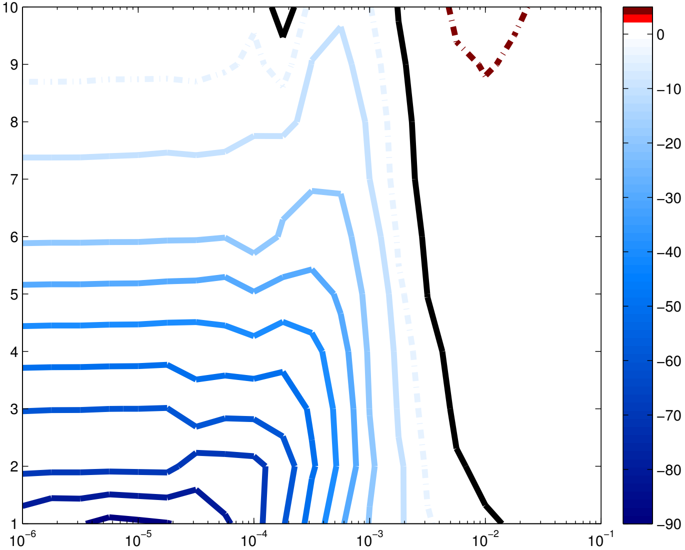
Data Profiles (vs random)



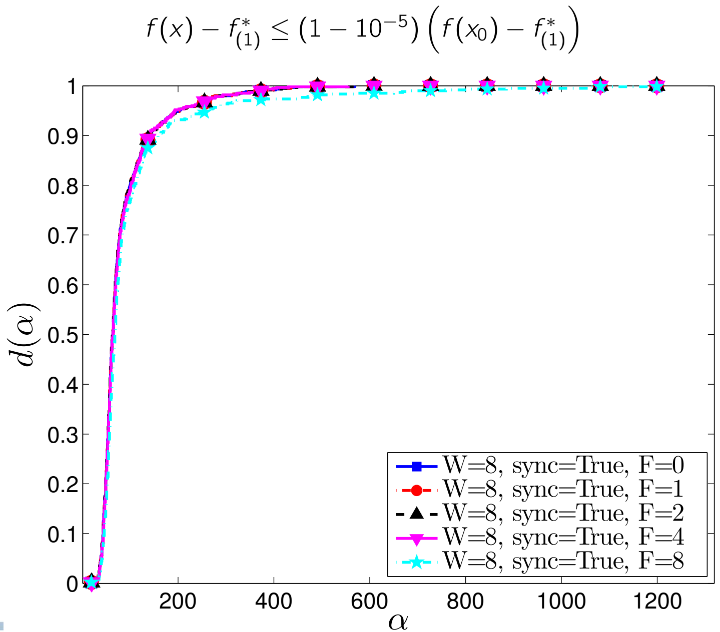
Data Profiles (vs random)



Data Profiles (vs random)

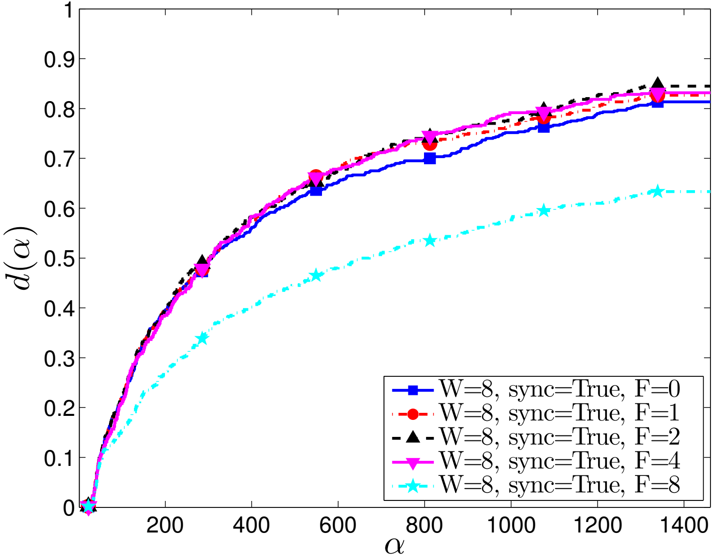


Data Profiles (forcing runs)



Data Profiles (forcing runs)

Within $\sqrt[n]{\frac{10^{-4}\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}$ of 3 best minima



Closing Remarks

- Concurrent function evaluations can locate multiple minima while efficiently finding a global minimum.

Closing Remarks

- Concurrent function evaluations can locate multiple minima while efficiently finding a global minimum.

- The ability to find many minima scales well with the number of workers.

Questions:

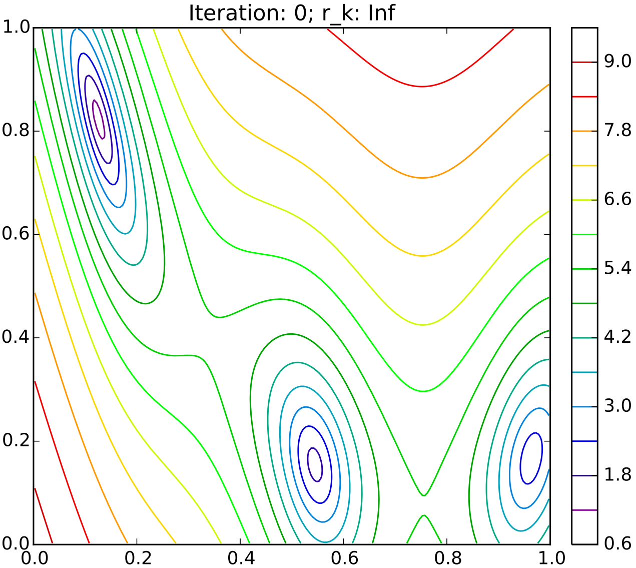
- Finding (or designing) the best local solver for our framework?
- Best way to process the queue?

(A)POSMM Manager

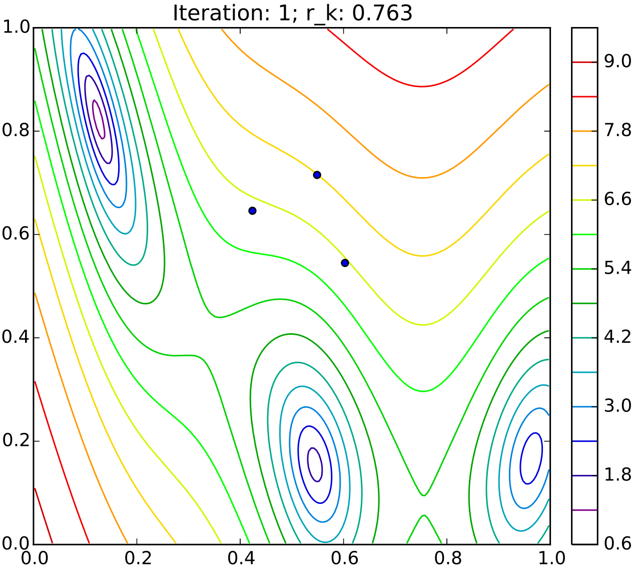
Algorithm 3: Manager Logic

while $k <$ fevalmax **do**
- MPI.recv(MPI.ANY_SOURCE)
- **if** *Received from Custodian* **then**
  - Check H_k
  - Add new point to Q_L
- **if** *Received from Worker* **then**
  - Update H_k
  - Possibly get a Custodian working on the next point
  - Run `decide_where_to_start`
  - Possibly update Q_L
- **if** sync=False *OR All Workers/Custodians are done* **then**
  - Give from Q_L or $\mathcal{R}_S$ to available worker(s)

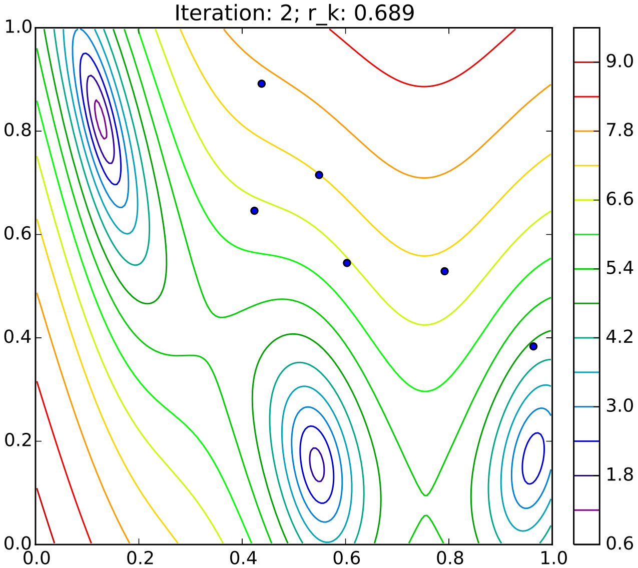
Pausing runs



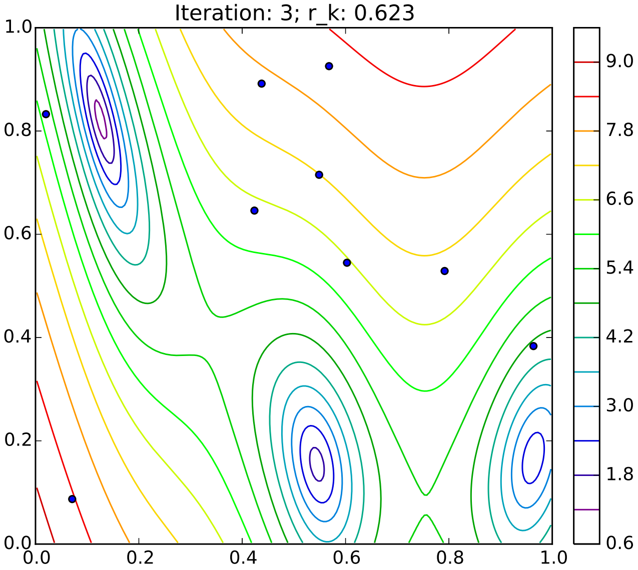
Pausing runs



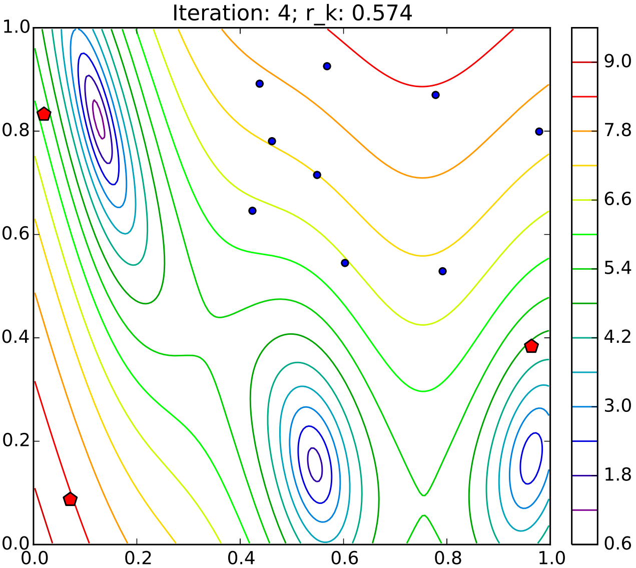
Pausing runs



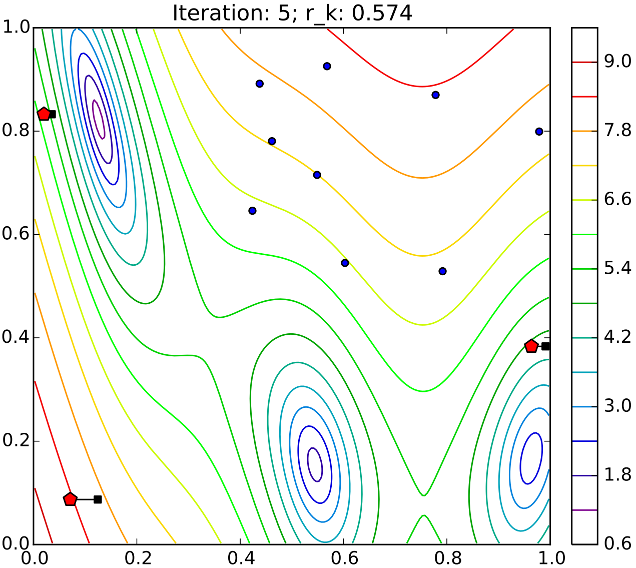
Pausing runs



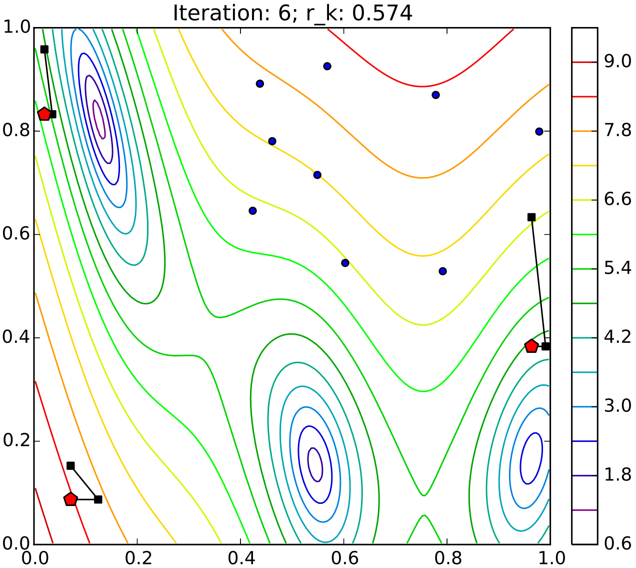
Pausing runs



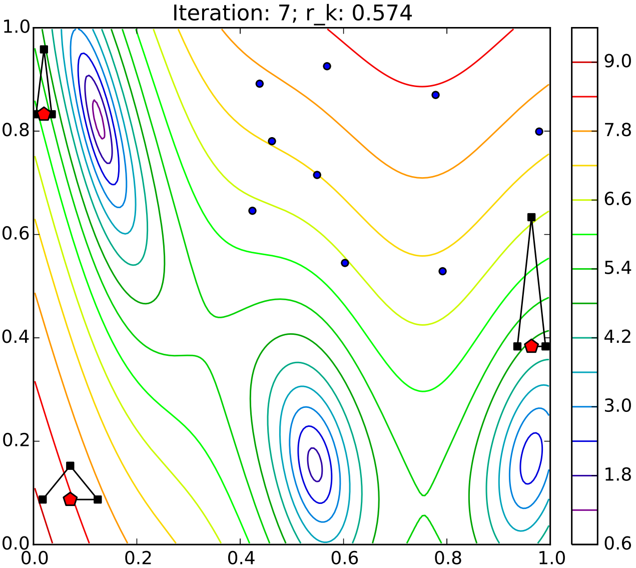
Pausing runs



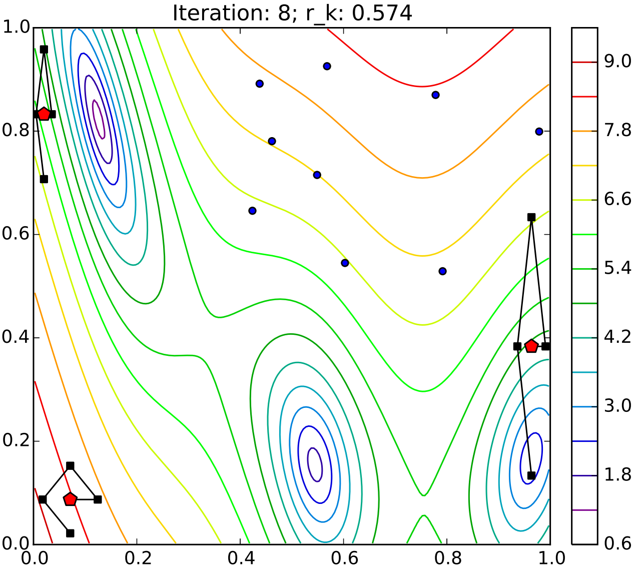
Pausing runs



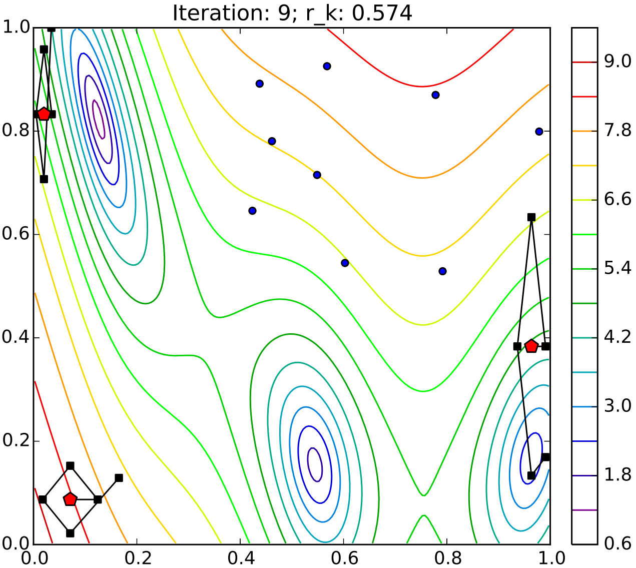
Pausing runs



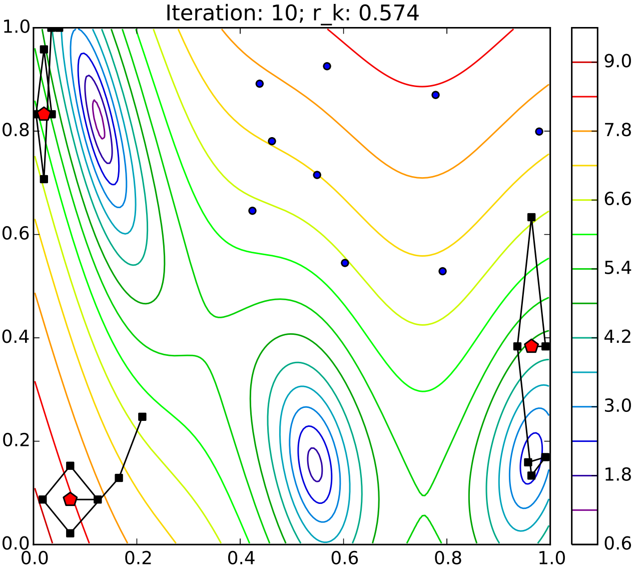
Pausing runs



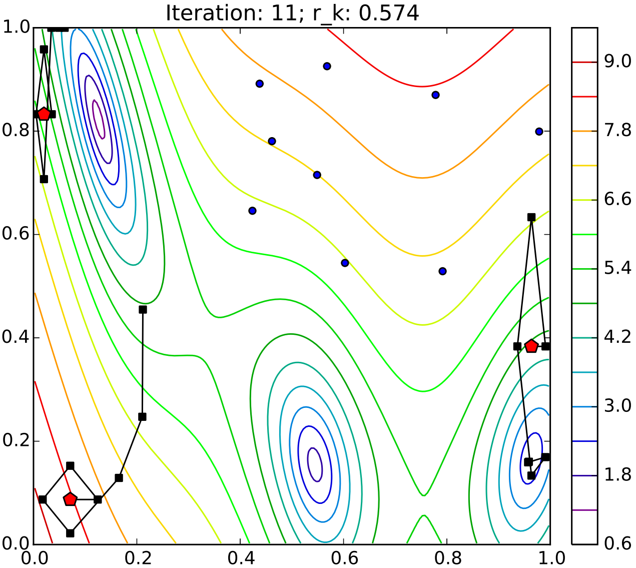
Pausing runs



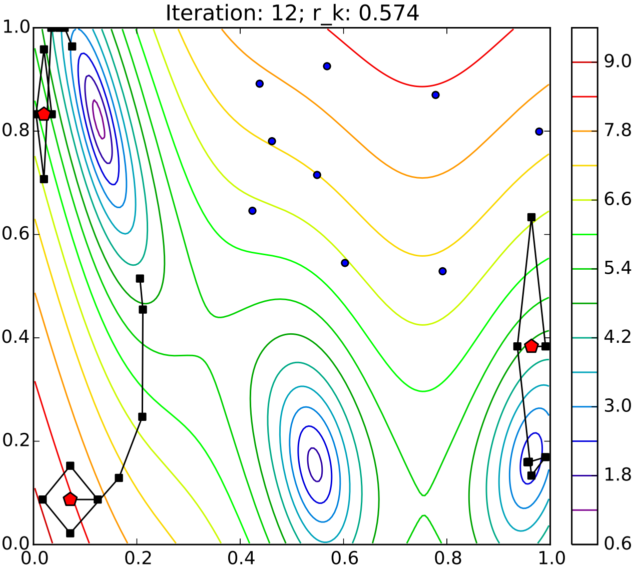
Pausing runs



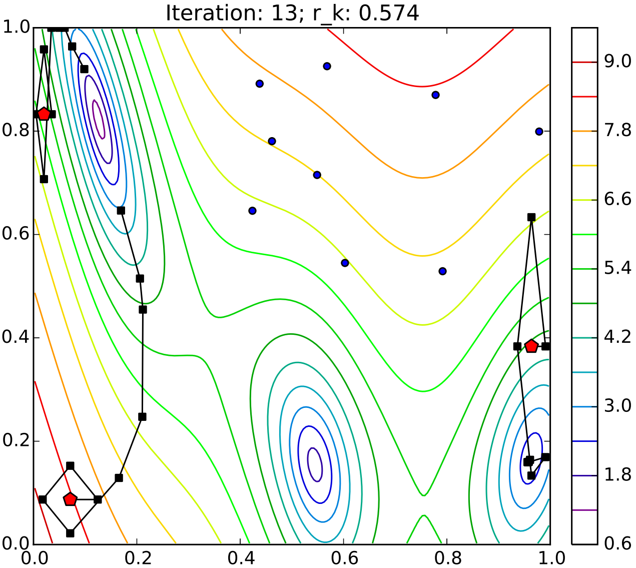
Pausing runs



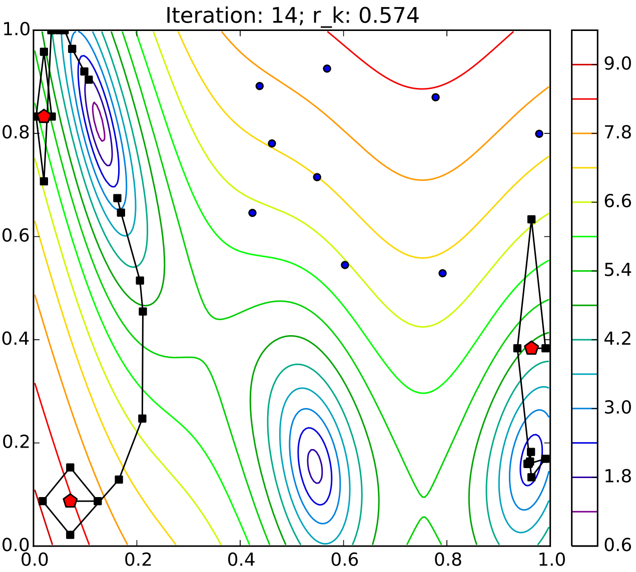
Pausing runs



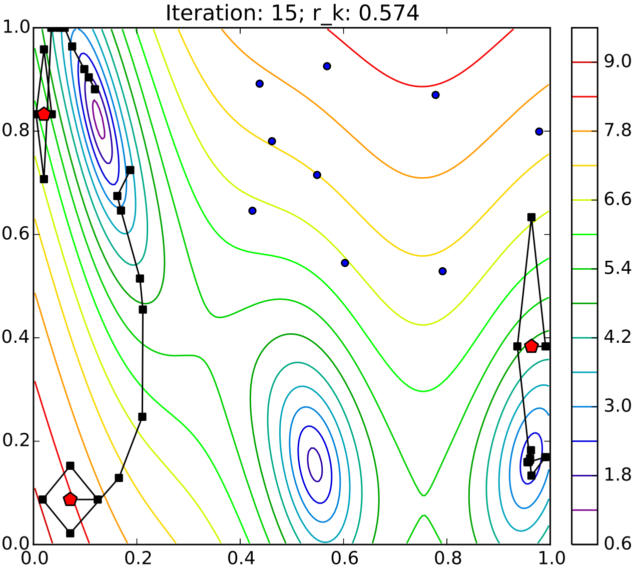
Pausing runs



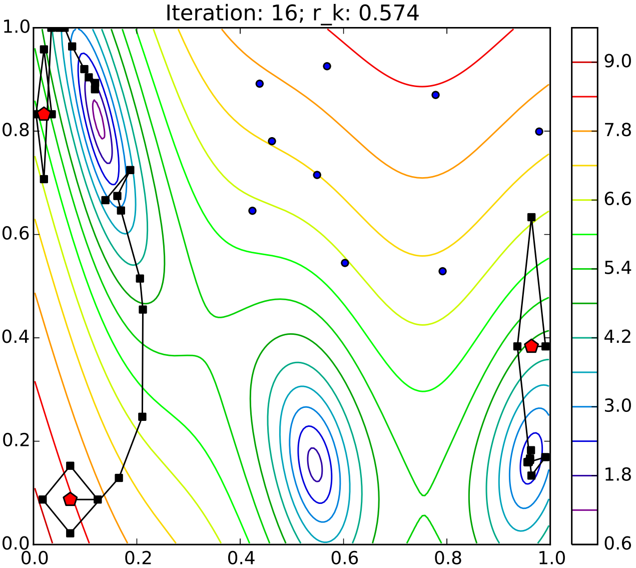
Pausing runs



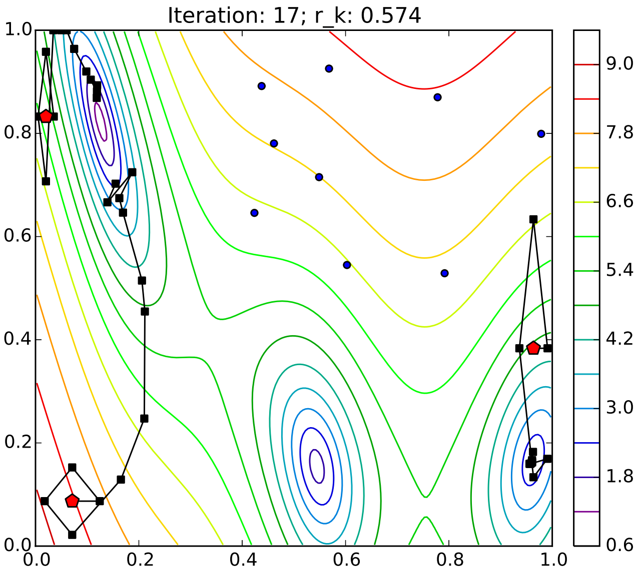
Pausing runs



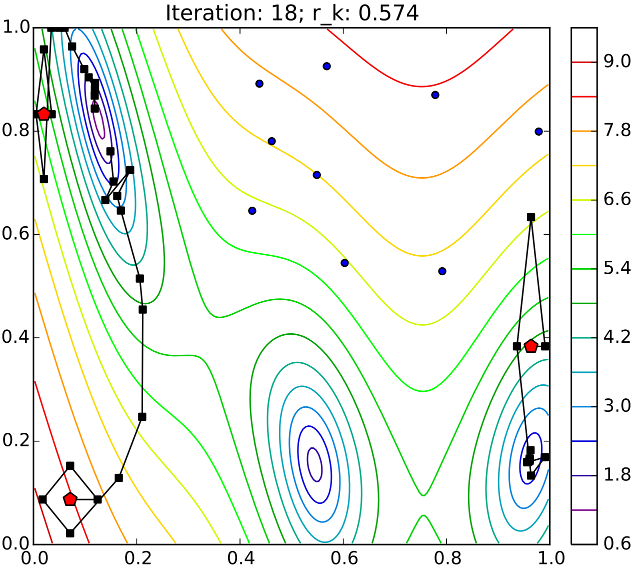
Pausing runs



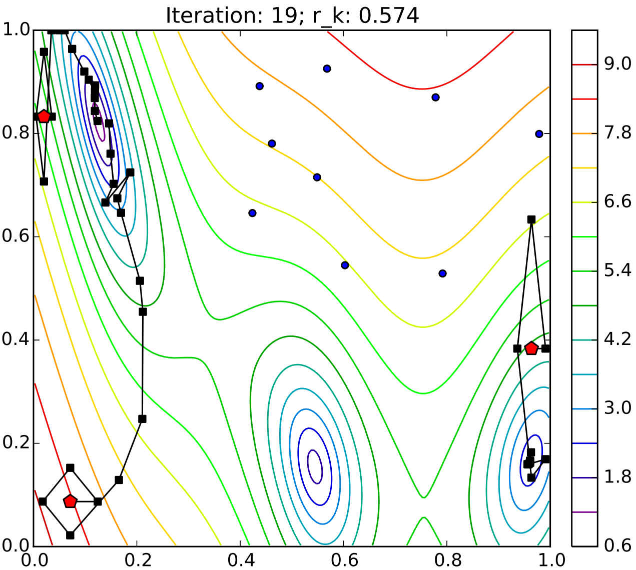
Pausing runs



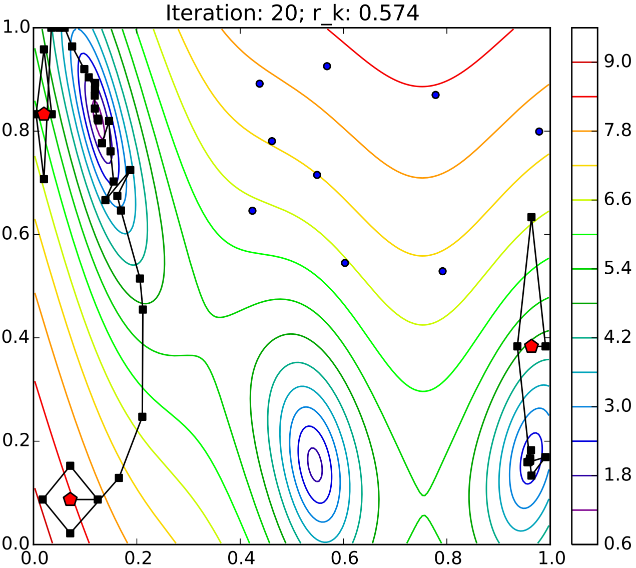
Pausing runs



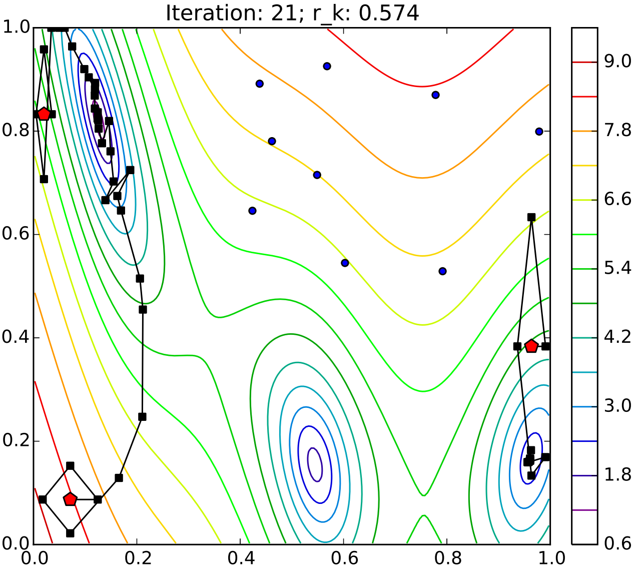
Pausing runs



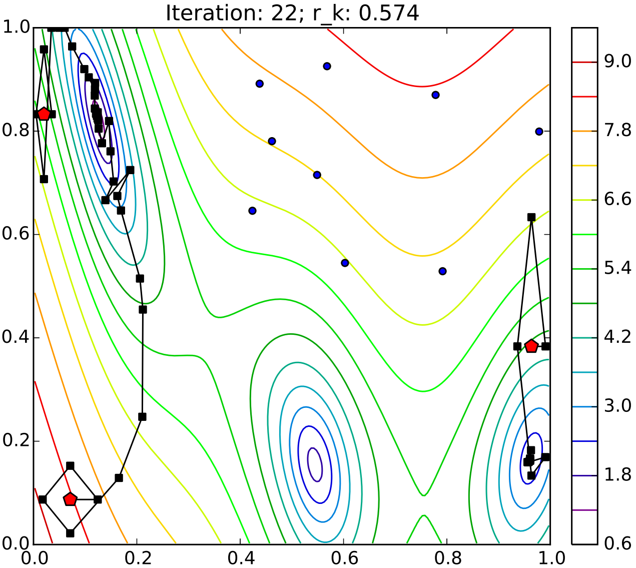
Pausing runs



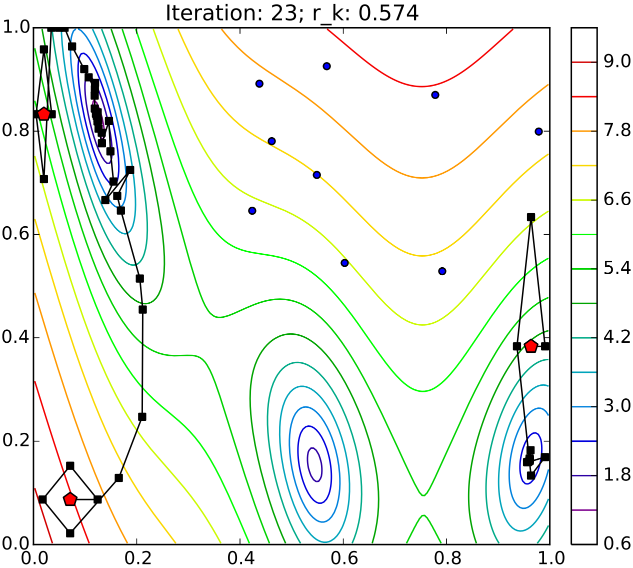
Pausing runs



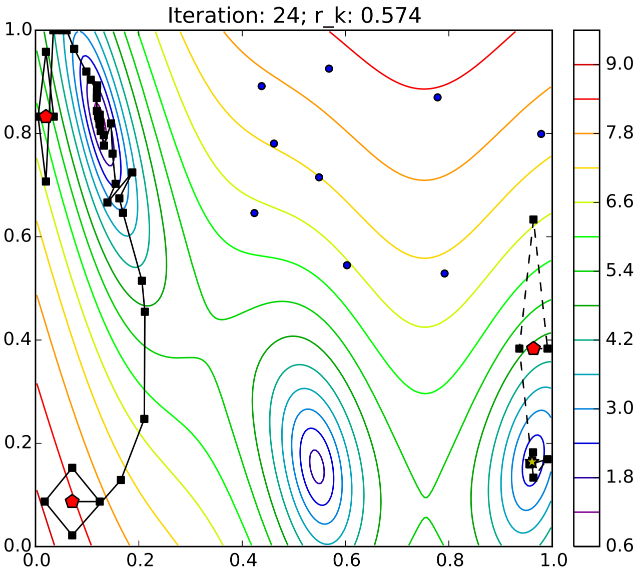
Pausing runs



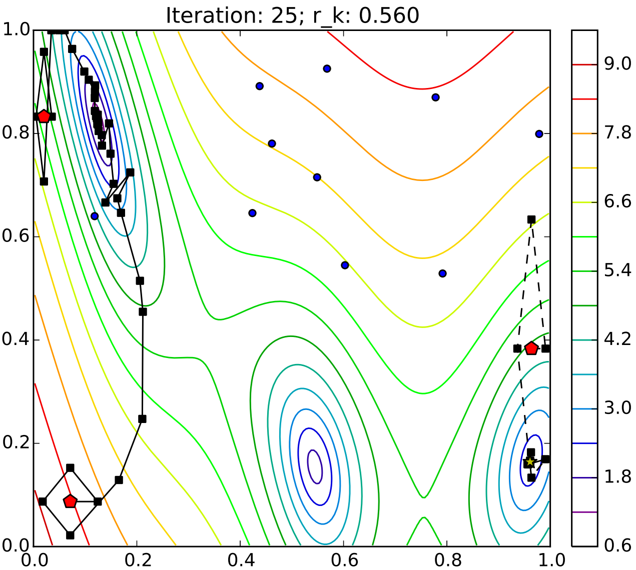
Pausing runs



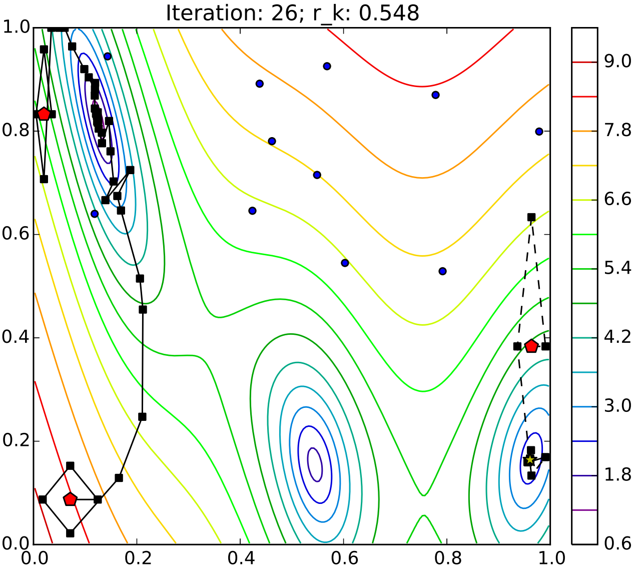
Pausing runs



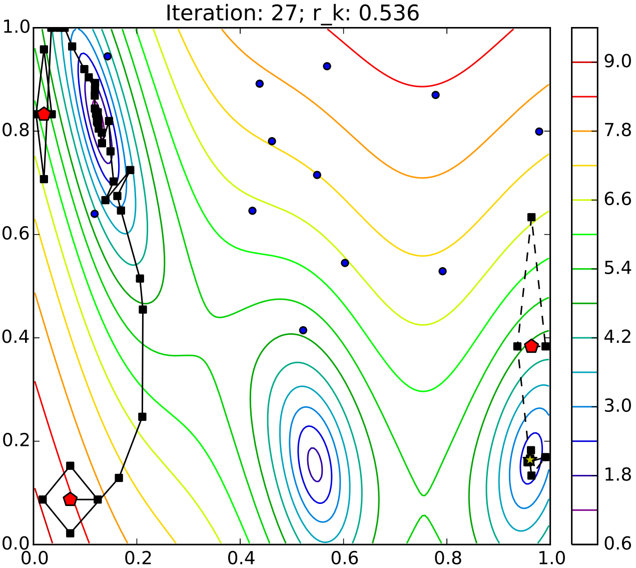
Pausing runs



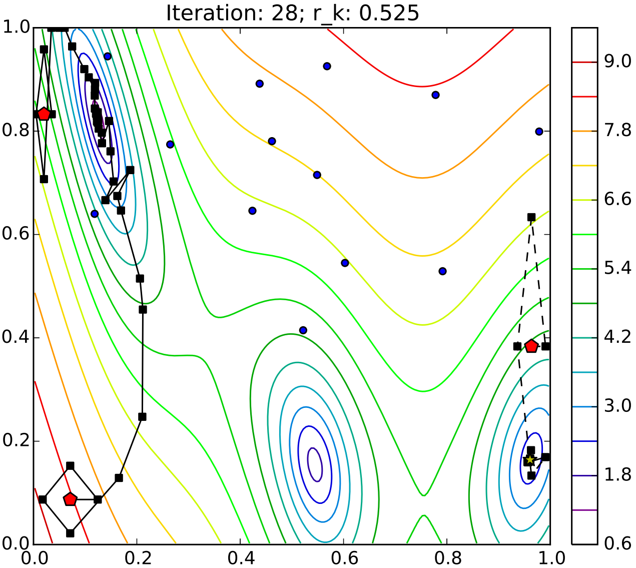
Pausing runs



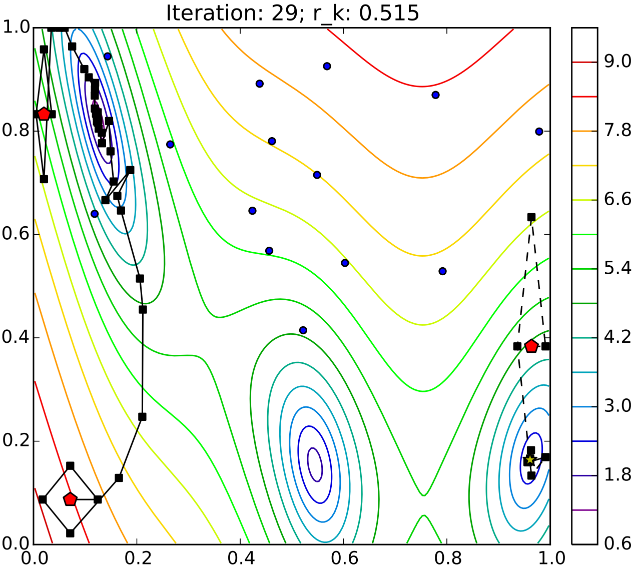
Pausing runs



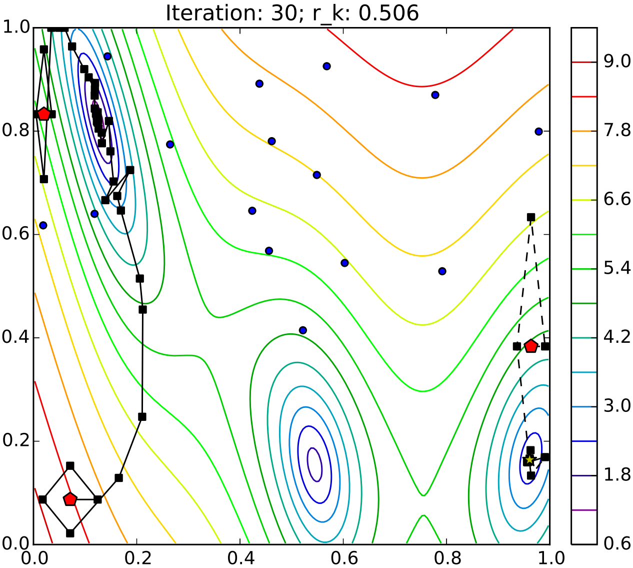
Pausing runs



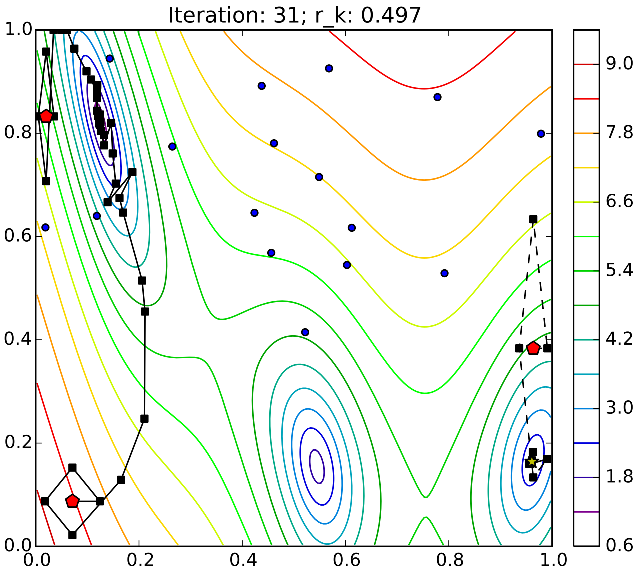
Pausing runs



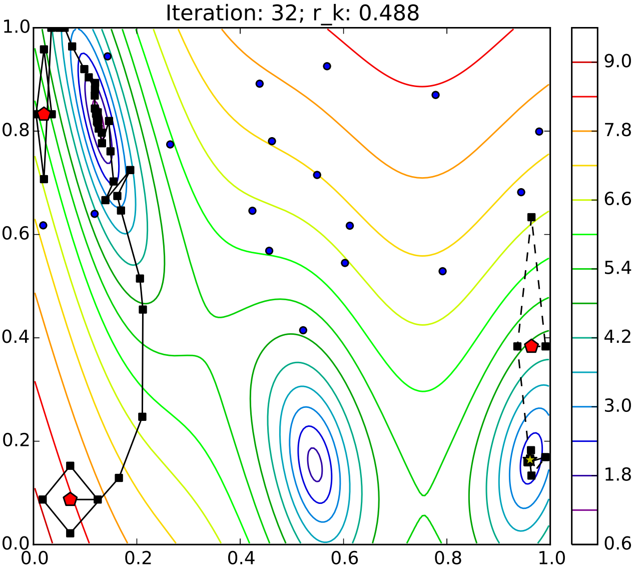
Pausing runs



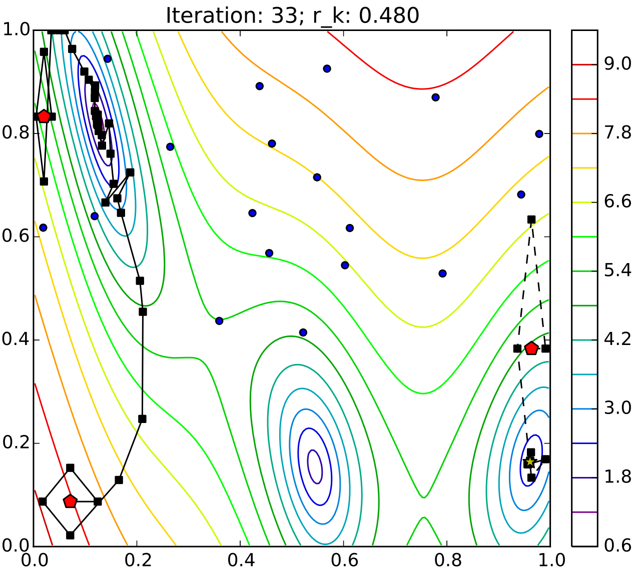
Pausing runs



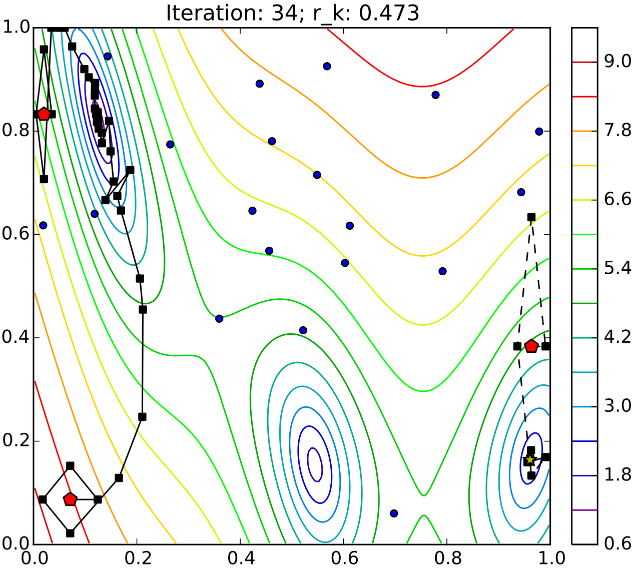
Pausing runs



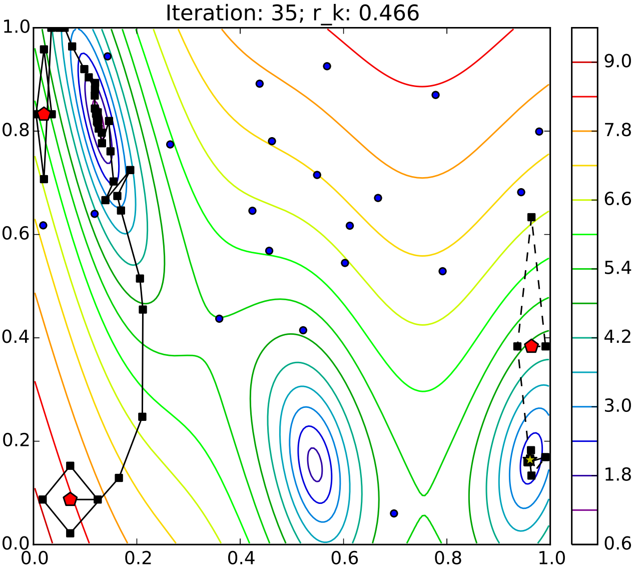
Pausing runs



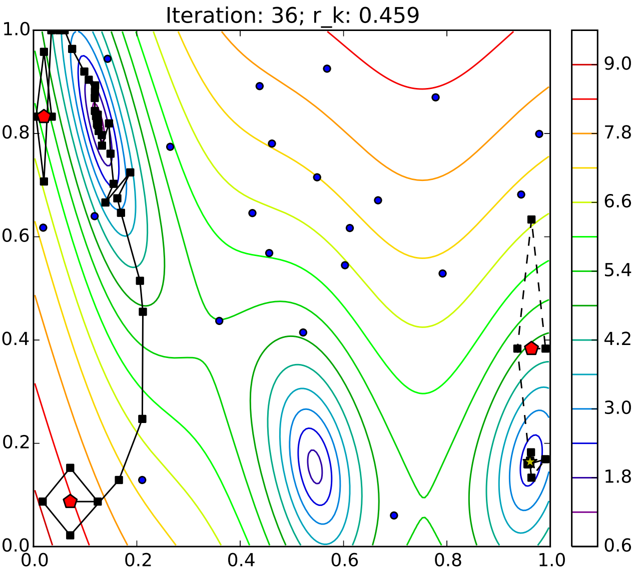
Pausing runs



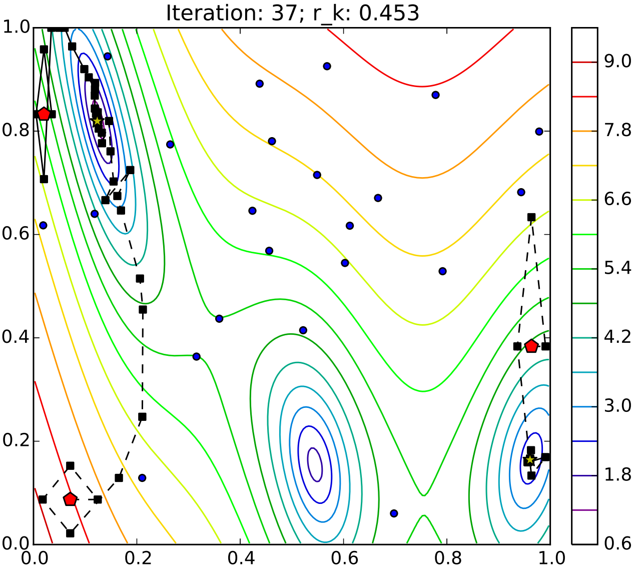
Pausing runs



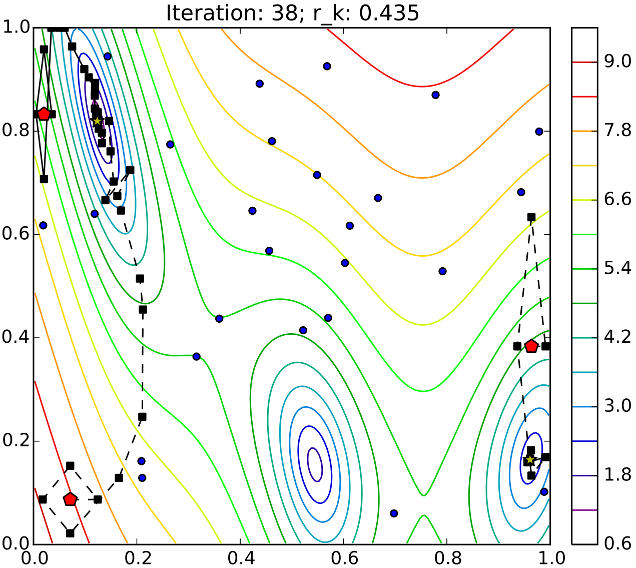
Pausing runs



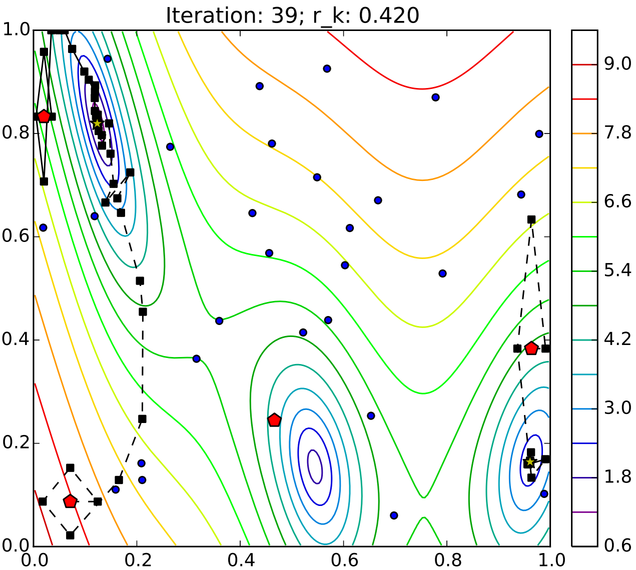
Pausing runs



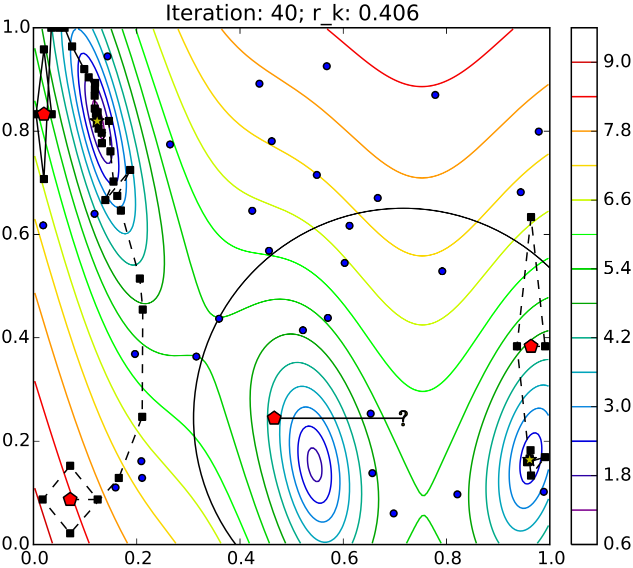
Pausing runs



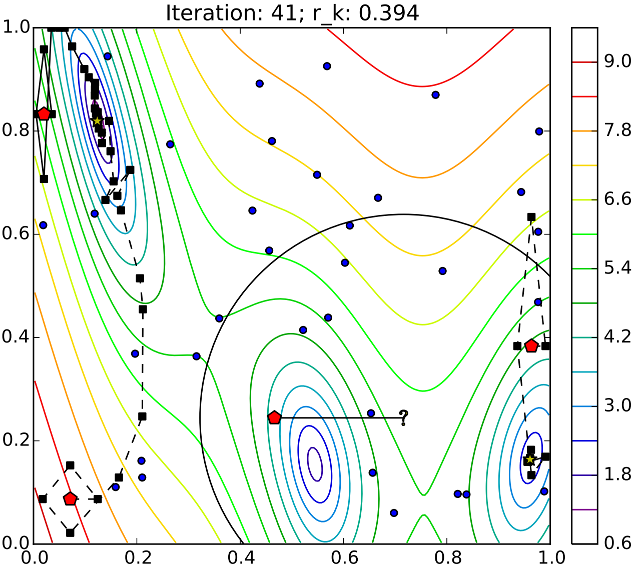
Pausing runs



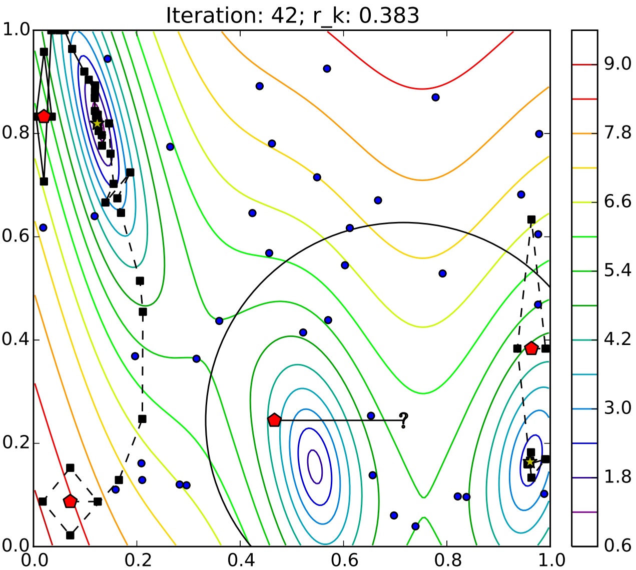
Pausing runs



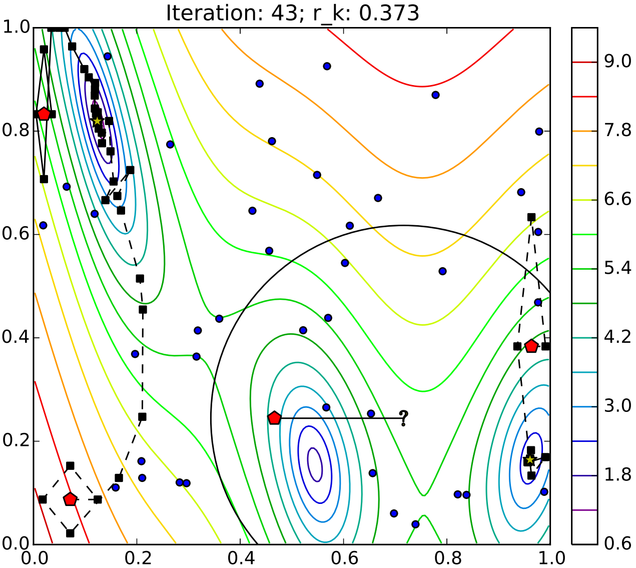
Pausing runs



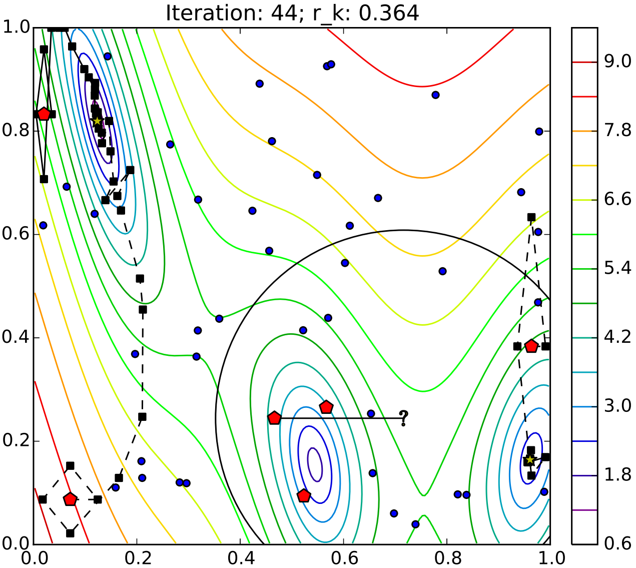
Pausing runs



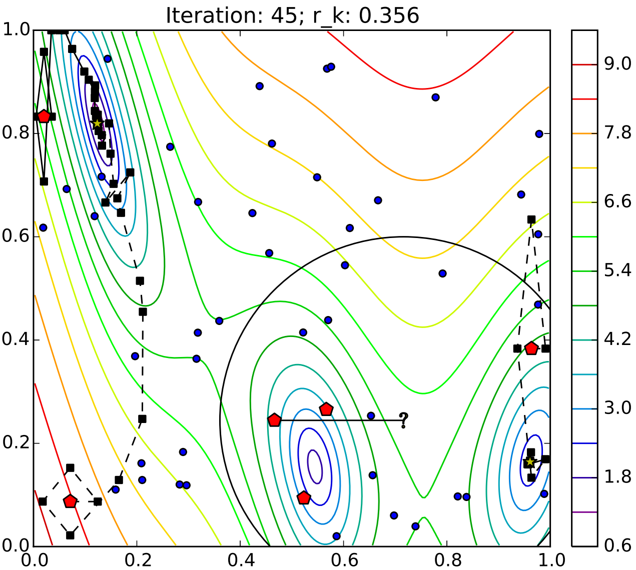
Pausing runs



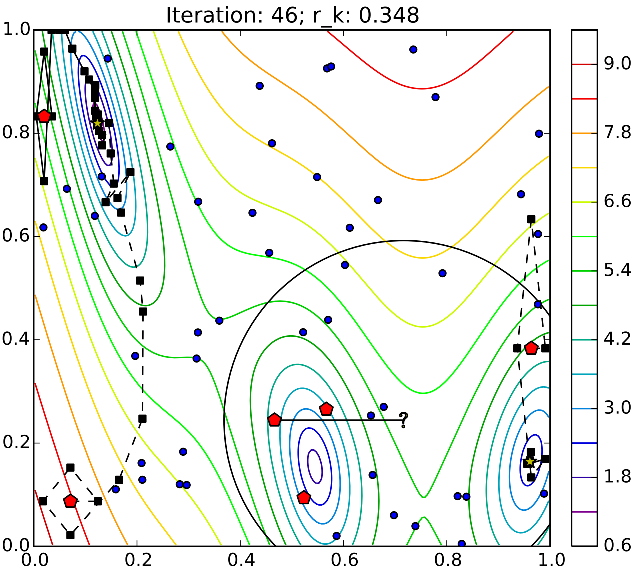
Pausing runs



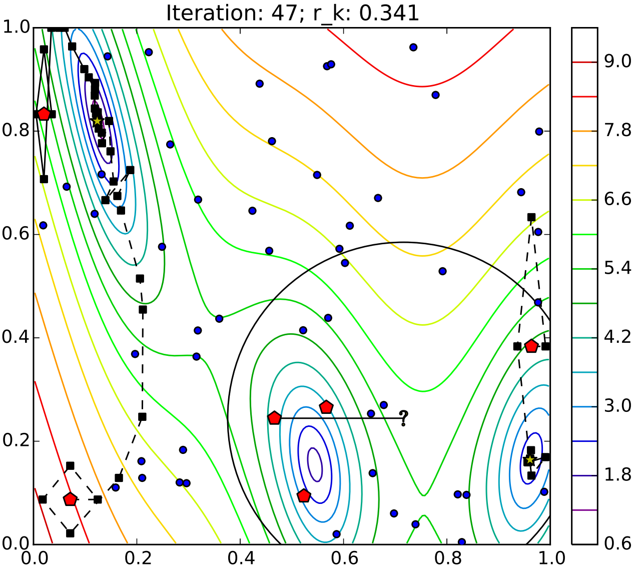
Pausing runs



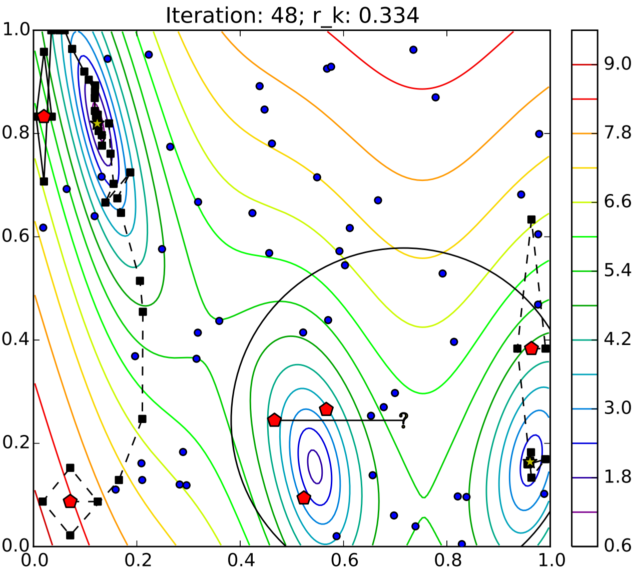
Pausing runs



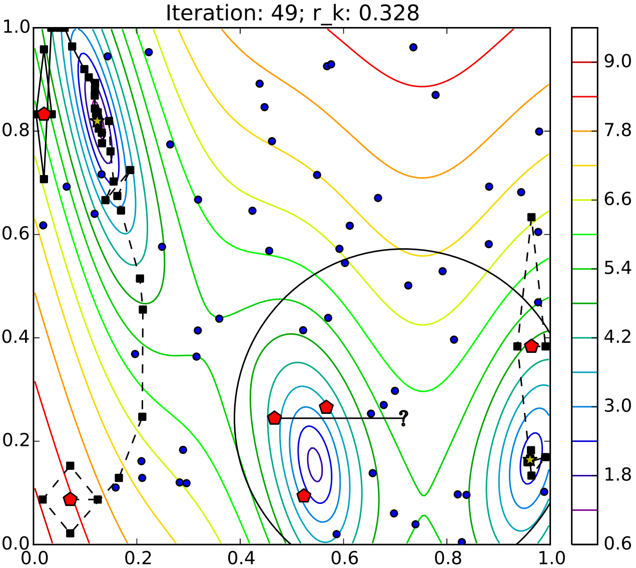
Pausing runs



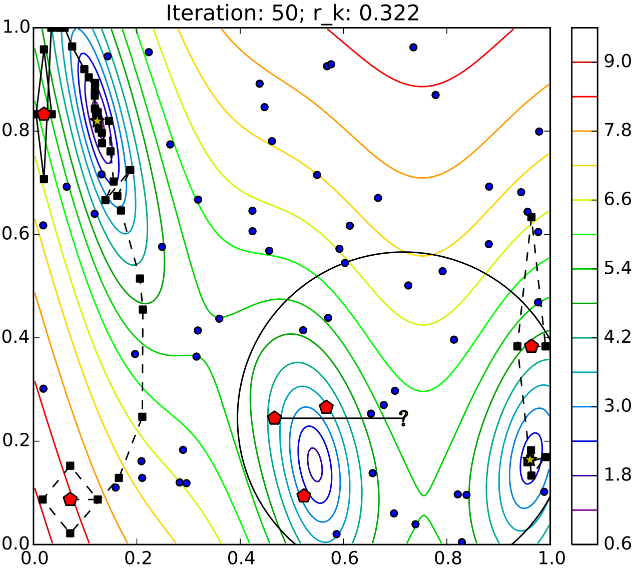
Pausing runs



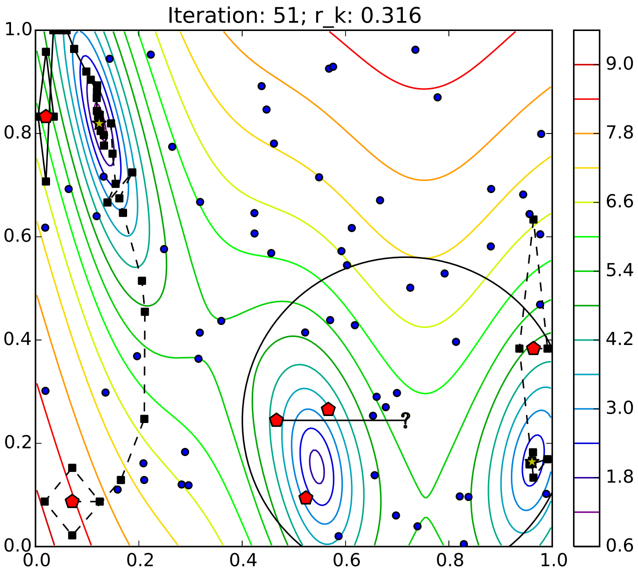
Pausing runs



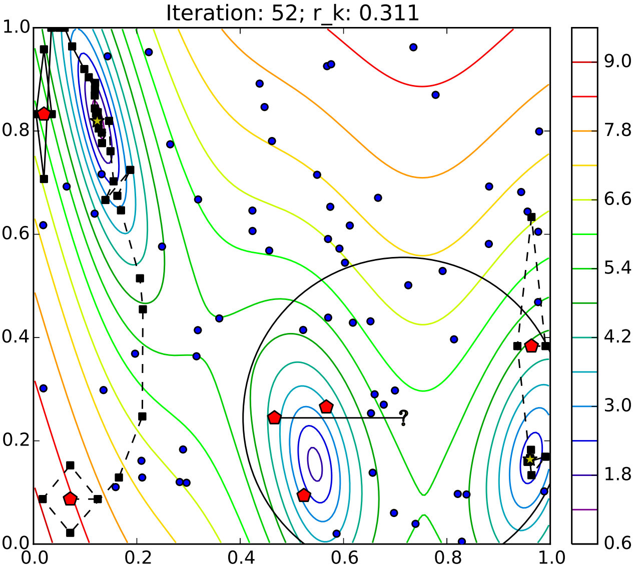
Pausing runs



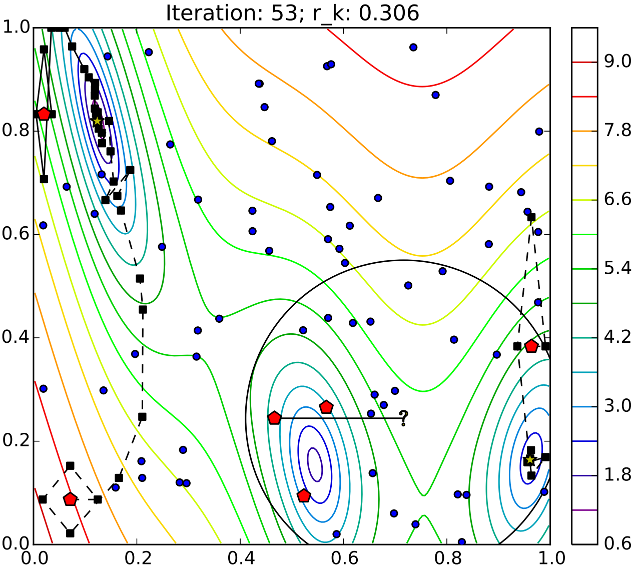
Pausing runs



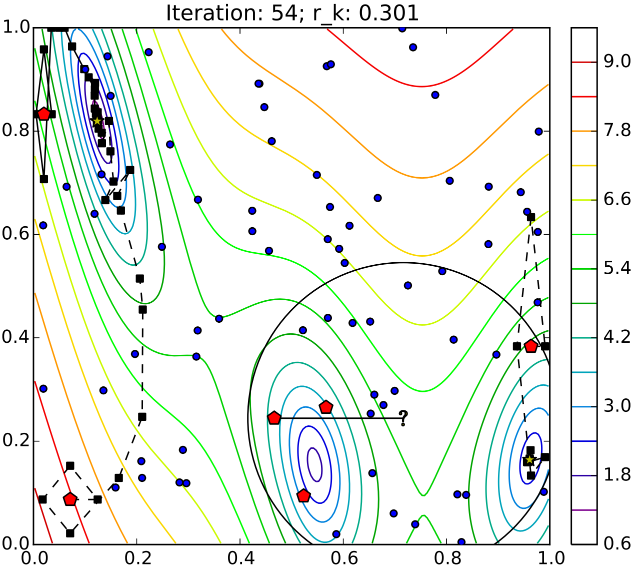
Pausing runs



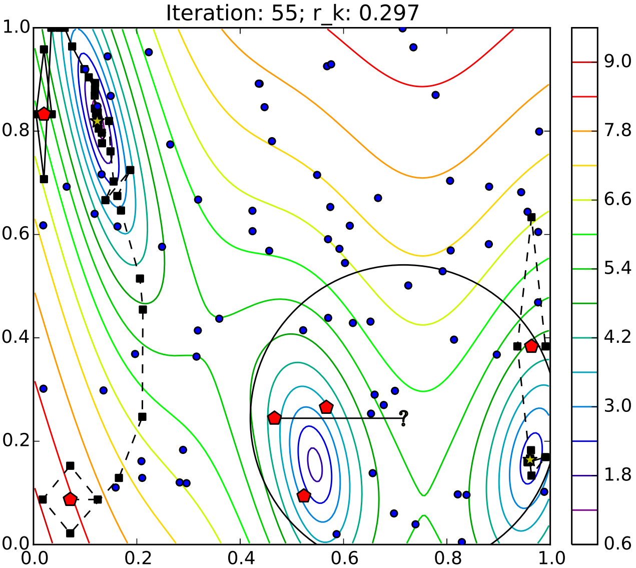
Pausing runs



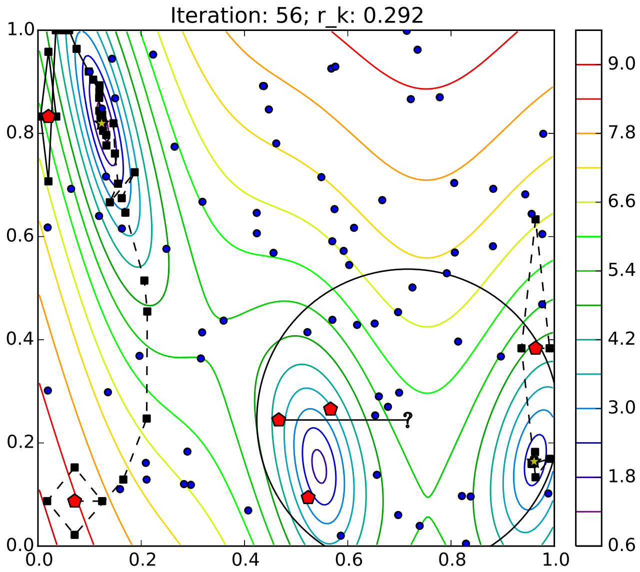
Pausing runs



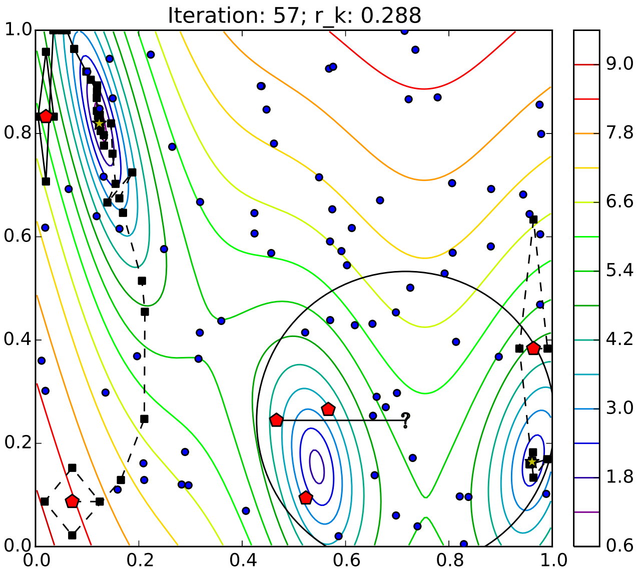
Pausing runs



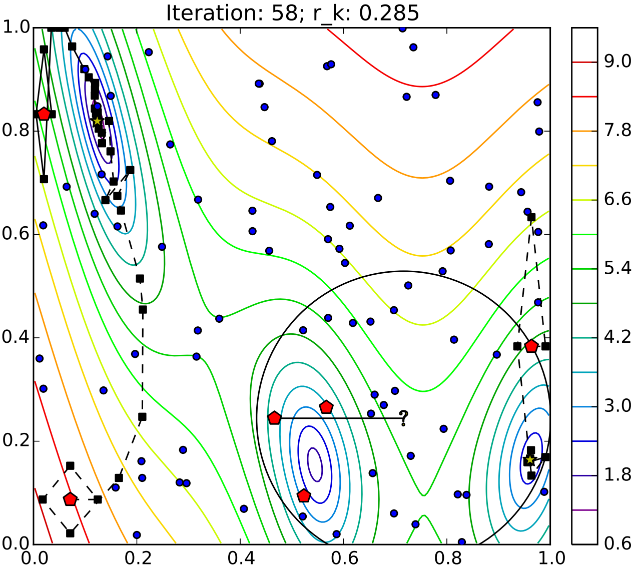
Pausing runs



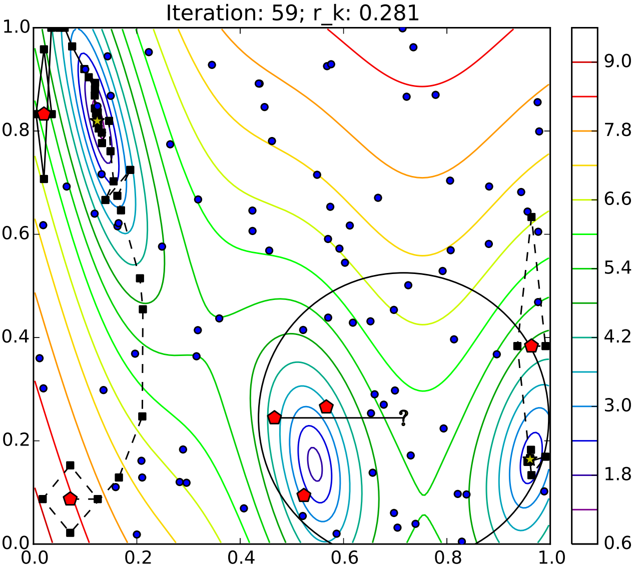
Pausing runs



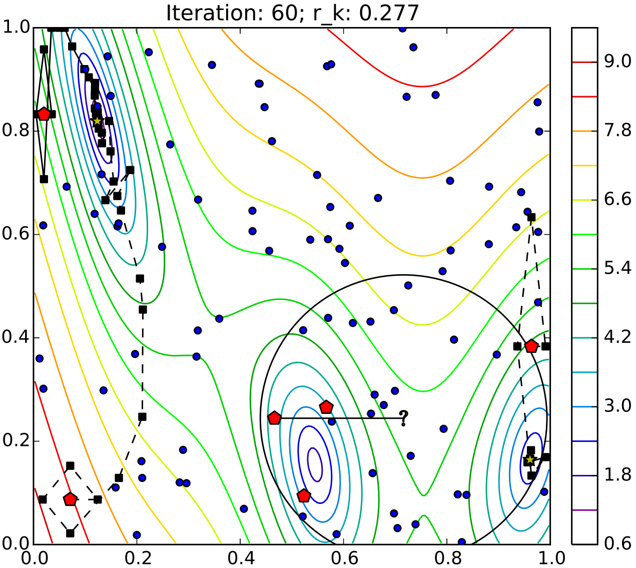
Pausing runs



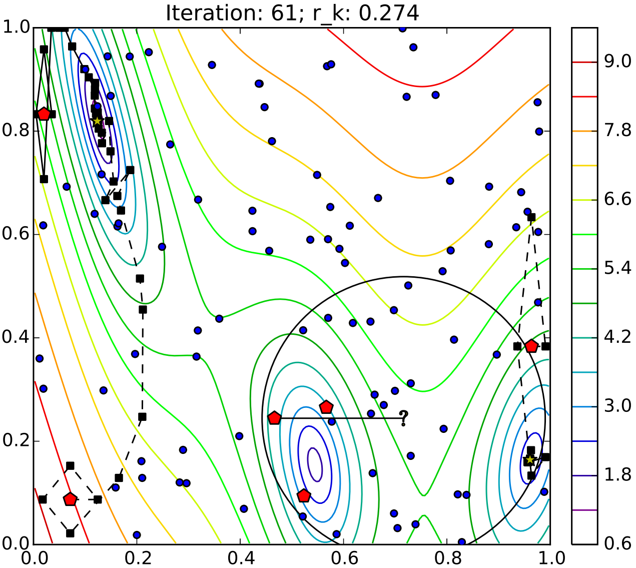
Pausing runs



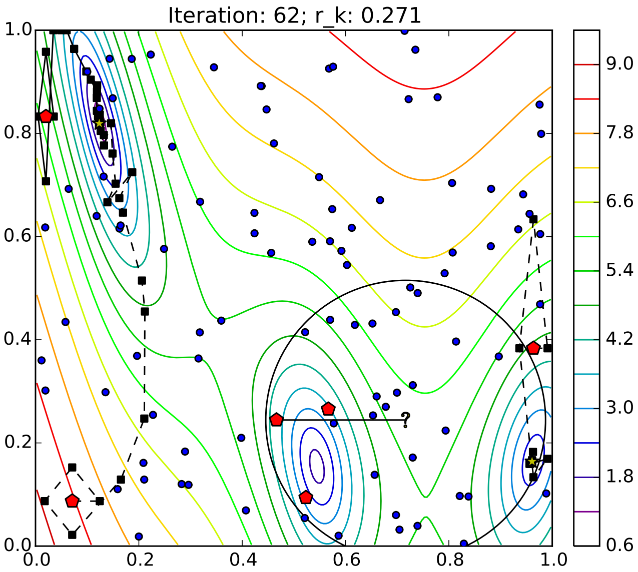
Pausing runs



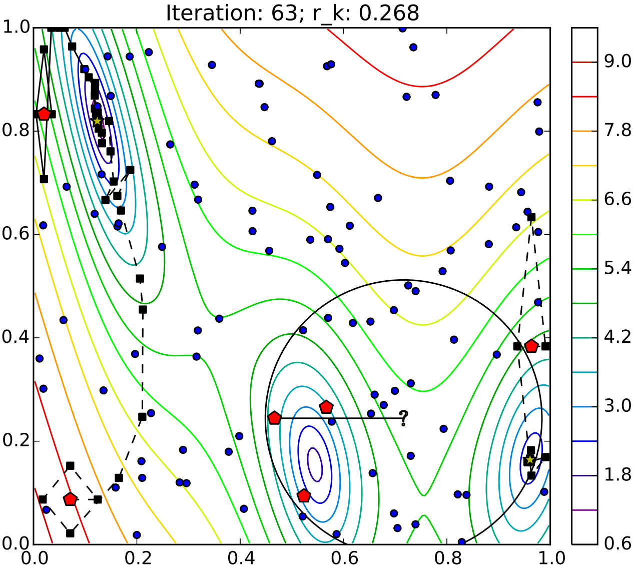
Pausing runs



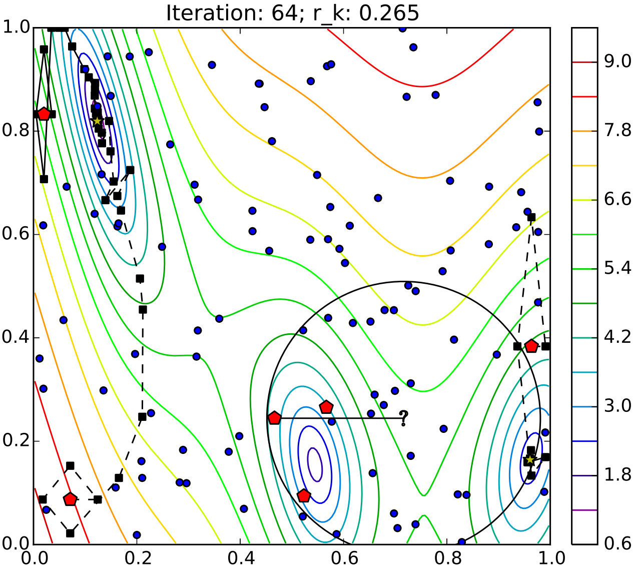
Pausing runs



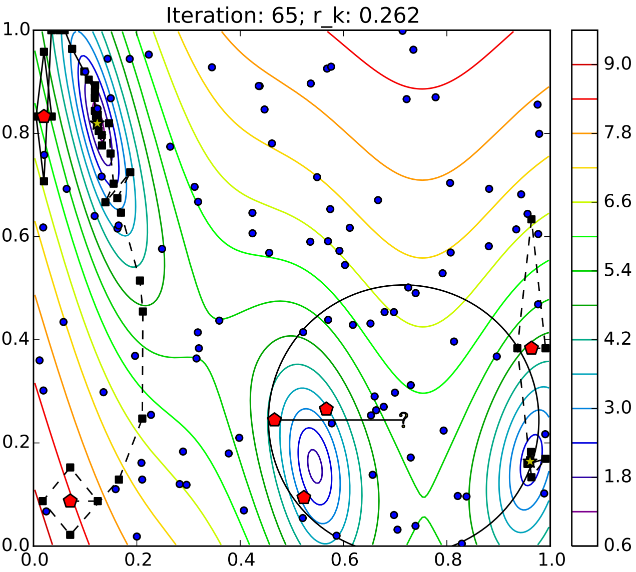
Pausing runs



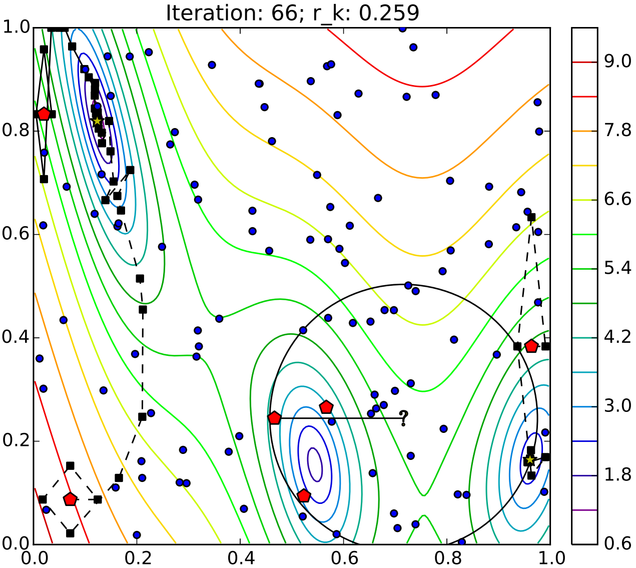
Pausing runs



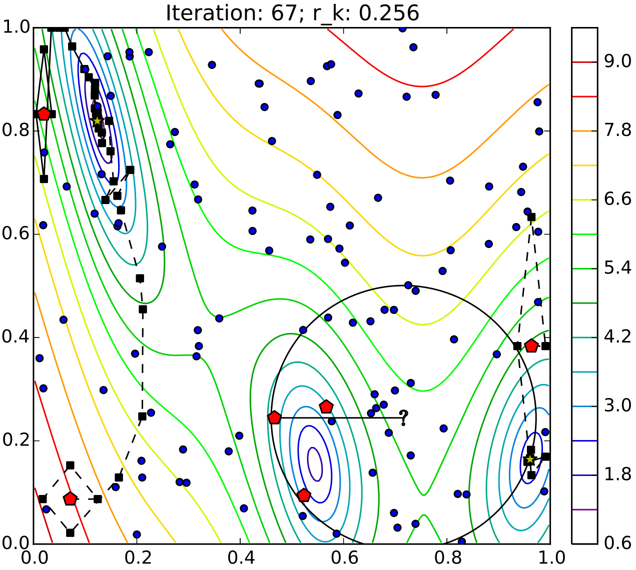
Pausing runs



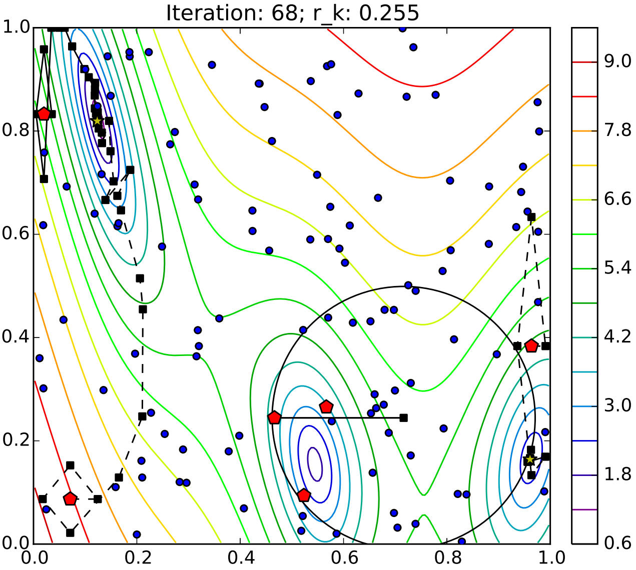
Pausing runs



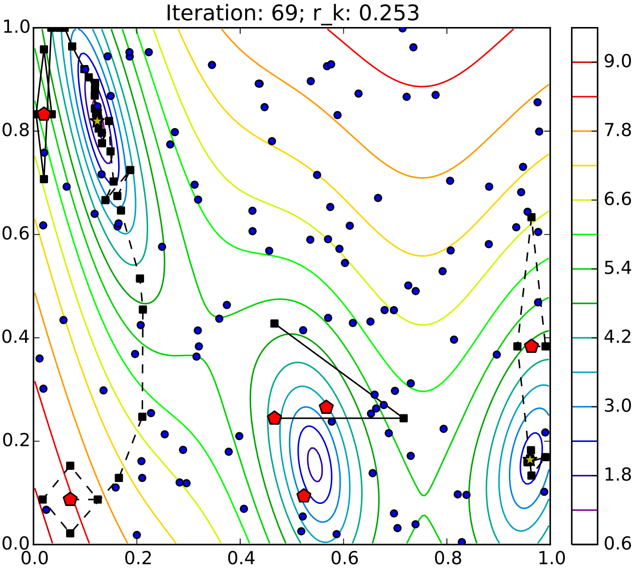
Pausing runs



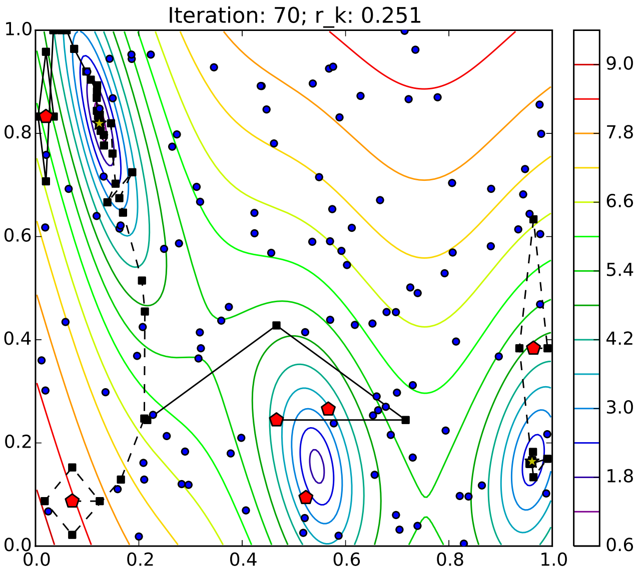
Pausing runs



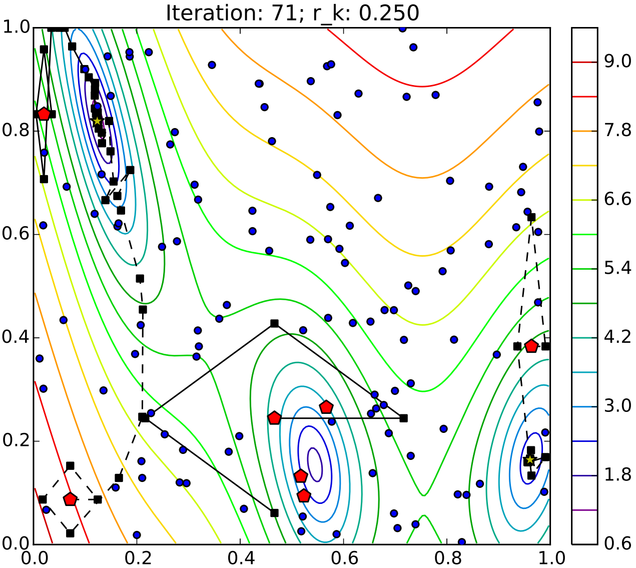
Pausing runs



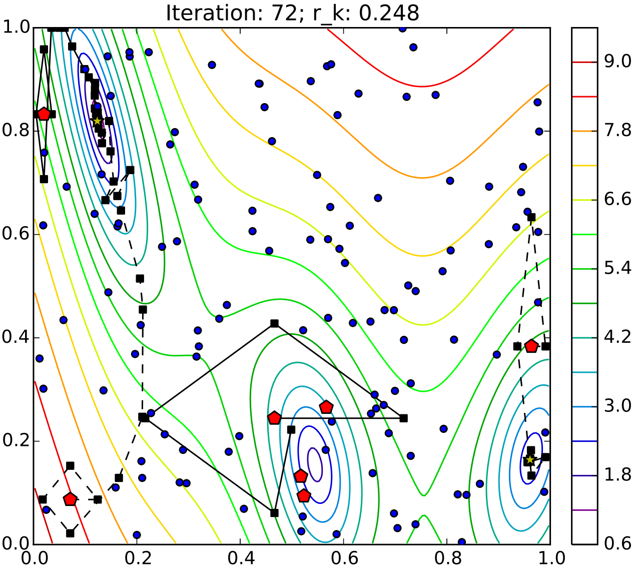
Pausing runs



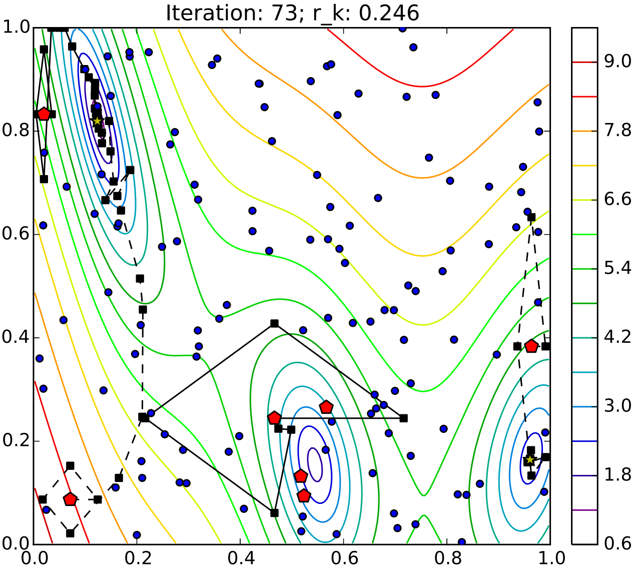
Pausing runs



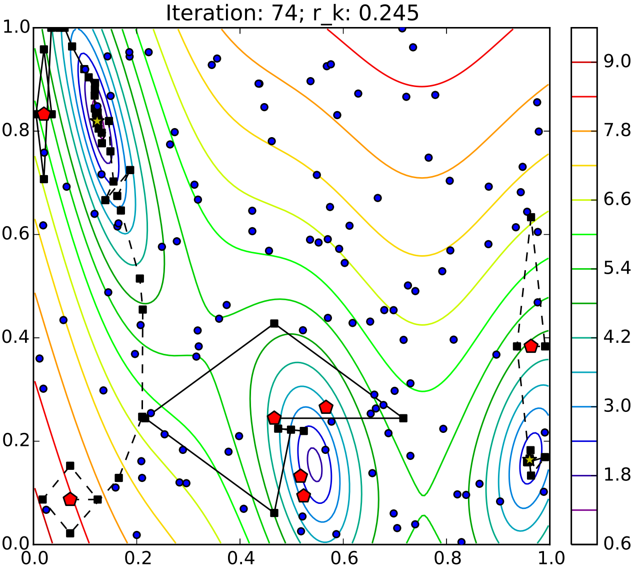
Pausing runs



Pausing runs



Pausing runs



Pausing runs